GT.0000027699

Lê Văn Phùng - Nguyễn Văn Tảo

Giáo trình
CÔNG NGHỆ PHẦN MỀM NÂNG CAO

NHÀ XUẤT BẢN
ĐẠI HỌC THÁI NGUYÊN

LÊ VĂN PHÙNG - NGUYỄN VĂN TẢO

GIÁO TRÌNH
CÔNG NGHỆ PHẦN MỀM
NÂNG CAO

NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN
NĂM 2018

MÃ SỐ: 04 - 105 ĐHTN - 2018

THUẬT NGỮ VÀ TỪ VIẾT TẮT

1. Tiếng Anh

4GT	forth generation technique	Kỹ thuật thế hệ 4
CASE	Computer Aided Software Engineering	Kỹ nghệ phần mềm được hỗ trợ bởi máy tính
CBSE	Component-Based Software Engineering	Kỹ nghệ phần mềm dựa trên thành phần
CMM	Capability Maturity Model	Mô hình trưởng thành
CMMI	Capability Maturity Model Intergration	Mô hình trưởng thành tích hợp
COM	Component Object Model	Mô hình đối tượng thành phần
CORBA	Common Object Request Broker Architecture	Một kiến trúc hướng đối tượng
DCOM	Distributed Component Object Model	Mô hình đối tượng phân tán
DSQI	Design Structure Quality Index	Chỉ số chất lượng về cấu trúc thiết kế
E_R	Entity-Relationship	Mô hình thực thể - mối quan hệ
EJB	Enterprise JavaBeans	Một mô hình thành phần phần mềm dành cho việc phát triển và triển khai các ứng dụng nghiệp vụ mức doanh nghiệp

HCI	Human - Computer Interface	Giao diện người - máy
HIPOC	Hierarch Input Process Output Chart	Sơ đồ phân cấp vào - ra
HTML	Hyper Text Markup Language	Ngôn ngữ đánh dấu siêu văn bản
HW	Hardware	Phần cứng
I-CASE	Integrated CASE	CASE tích hợp
IDL	Interface Description Language	Ngôn ngữ mô tả giao diện
IEEE	Institute Electrical and Electronic Engineers	Viện kỹ nghệ điện và điện tử
ISO	International Standards Organization	Tổ chức quốc tế về chuẩn hóa
IT	Information Technology	Công nghệ thông tin
LAN	Local Arear Network	Mạng cục bộ
NF	Normal form	Dạng chuẩn
OMG	Object Management Group	Nhóm quản lý đối tượng
OOA	Object - Oriented Approach	Cách tiếp cận hướng đối tượng
OOP	Object - Oriented Programming	Phương thức lập trình hướng đối tượng
ORB	Object Request Broker	Bộ phận trừu tượng trung gian
PC	Personal Computer	Máy tính cá nhân

RAD	Rapid Application Development	Phát triển ứng dụng nhanh
RUP	Rational Unified Process	Tiến trình thống nhất
SC	Software configuration	Cấu hình phần mềm
SCI	Software Configuration Item	Khoản mục cấu hình phần mềm
SCM	Software Configuration Management	Quản lý cấu hình phần mềm
SEI	Software Engineering Institute	Viện công nghệ phần mềm Mỹ
SMI	Software Maturity Index	Chỉ số trưởng thành phần mềm
SOA	Service Oriented Architecture	Kiến trúc hướng dịch vụ
SOAP	Simple Object Access Protocol	Giao thức truy xuất đối tượng đơn giản
SQA	Software Quality Assurance	Bảo đảm chất lượng phần mềm
SW	Software	Phần mềm
UDDI	Universal Description, Discovery and Integration	Một thành phần đăng ký dịch vụ
UIDS	User interface development system	Hệ thống phát triển giao diện người dùng
UML	Unified Modeling Language	Ngôn ngữ mô hình hợp nhất

VORD	Viewpoint-Oriented Requirements Definition	Xác định yêu cầu hướng theo cách nhìn
WS	Web Service	Dịch vụ web
WSDL	Web Services Description Language	Ngôn ngữ đặc tả dịch vụ web
XML	Extensible Makup Language	Ngôn ngữ đánh dấu mở rộng

2. Tiếng Việt

CNTT Công nghệ Thông tin

CSDL Cơ sở dữ liệu

HTTT Hệ thống thông tin

NSD Người sử dụng

MỤC LỤC

Trang

THUẬT NGỮ VÀ TỪ VIẾT TẮT ... 3

LỜI NÓI ĐẦU ... 15

Chương 1. TỔNG QUAN VỀ CÔNG NGHỆ PHẦN MỀM ... 18

1.1. Định nghĩa phần mềm ... 18

1.2. Định nghĩa công nghệ phần mềm ... 19

1.3. Sự khác biệt giữa công nghệ phần mềm với kỹ nghệ phần mềm ... 20

1.4. Sự khác biệt giữa kỹ nghệ phần mềm và kỹ nghệ hệ thống ... 20

1.5. Các phương pháp trong kỹ nghệ phần mềm ... 22

1.6. Bàn thợ CASE ... 24

1.6.1. Khái niệm về CASE ... 25

1.6.2. Các loại công cụ CASE ... 26

1.6.3. Môi trường CASE tích hợp ... 28

1.7. Tiêu chuẩn của một phần mềm tốt ... 31

1.8. Những thách thức chính đối với công nghệ phần mềm ... 32

1.9. Tổng quan một số hướng kỹ nghệ phần mềm điển hình ... 33

1.9.1. Kỹ nghệ hướng cấu trúc ... 33

1.9.2. Kỹ nghệ hướng đối tượng ... 35

1.9.3. Kỹ nghệ hướng thành phần ... 37

1.9.4. Kỹ nghệ hướng dịch vụ ... 41

1.9.5. Điện toán đám mây ... 43

TÓM TẮT CHƯƠNG ... 46

CÂU HỎI CHƯƠNG 1 ... 48

BÀI TẬP CHƯƠNG 1 ... 48

Chương 2. TIẾN TRÌNH PHÁT TRIỂN PHẦN MỀM 50
2.1. Tiến trình phần mềm 50
2.1.1. Khái niệm tiến trình phần mềm 50
2.1.2. Khái niệm mô hình tiến trình 51
2.2.Mô hình thác nước 51
2.3. Mô hình tiến hóa 52
2.3.1. Mô hình làm bản mẫu 53
2.3.2. Mô hình xoắn ốc 54
2.3.3. Mô hình RUP 56
2.3.4. Mô hình phát triển tăng dần 56
2.3.5. Mô hình phát triển ứng dụng nhanh RAD 57
2.3.6. Mô hình phát triển hệ thống hình thức hóa 57
2.3.7. Mô hình phát triển phần mềm theo hướng sử dụng lại 58
2.3.8. Mô hình phát triển phần mềm theo kỹ thuật thế hệ thứ tư 59
2.4. Các hoạt động của tiến trình phát triển phần mềm 60
TÓM TẮT CHƯƠNG 63
CÂU HỎI CHƯƠNG 2 64
Chương 3. ĐẶC TẢ YÊU CẦU PHẦN MỀM VÀ YÊU CẦU HỆ THỐNG . 65
3.1. Các yêu cầu hệ thống 65
3.1.1. Khái niệm về yêu cầu hệ thống 65
3.1.2. Phân loại yêu cầu hệ thống 66
3.1.3. Các cách tiếp cận và các kỹ thuật phân tích yêu cầu hệ thống 68
3.2. Các yêu cầu phần mềm 72
3.3. Tài liệu đặc tả yêu cầu 75
3.4. Tiến trình xác định yêu cầu và phát triển bản mẫu 81
3.4.1. Tiến trình xác định và đặc tả yêu cầu phần mềm 81
3.4.2. Tiến trình phát triển bản mẫu hệ thống 83
3.4.3. Tiến trình phát triển bản mẫu phần mềm 85

TÓM TẮT CHƯƠNG 87
CÂU HỎI CHƯƠNG 3 89
BÀI TẬP CHƯƠNG 3 89
Chương 4. ĐẶC TẢ THIẾT KẾ 91
4.1. Thiết kế và thiết kế phần mềm 91
4.2. Thiết kế dữ liệu 94
4.2.1. Thiết kế cơ sở dữ liệu mức logic 94
4.2.2. Thiết kế cơ sở dữ liệu mức vật lý 95
4.3. Thiết kế cấu trúc chương trình 96
4.4. Thiết kế giao diện 97
4.4.1. Ý nghĩa và bản chất của thiết kế giao diện người- máy 97
4.4.2. Sơ đồ tương tác hệ thống giao diện 98
4.4.3. Cấu trúc tương tác trên màn hình 100
4.4.4. Những nguyên tắc thiết kế 101
4.4.5. Những yêu cầu chung đối với thiết kế giao diện 102
4.4.6. Tiến trình thiết kế giao diện 104
4.5. Thiết kế mô-đun xử lý 105
4.5.1. Cách tiếp cận top-down 105
4.5.2. Phương pháp tinh chỉnh từng bước 106
4.5.3.Đặc tả mô-đun 107
TÓM TẮT CHƯƠNG 109
CÂU HỎI CHƯƠNG 4 111
BÀI TẬP CHƯƠNG 4 112
Chương 5. ĐẶC TẢ THIẾT KẾ KIẾN TRÚC 113
5.1. Kiến trúc phần mềm và đặc tả thiết kế kiến trúc 113
5.1.1. Định nghĩa kiến trúc phần mềm 113
5.1.2. Vai trò và tầm quan trọng của kiến trúc phần mềm 116
5.1.3. Đặc tả thiết kế kiến trúc 116

5.2. Các mô hình kiến trúc ứng dụng 119
5.2.1. Kiến trúc phần mềm theo hướng dữ liệu tập trung hay phân tán .. 120
5.2.2. Kiến trúc phần mềm theo hướng điều khiển tập trung 124
5.2.3. Kiến trúc phần mềm theo hướng sự kiện 125
5.2.4. Kiến trúc phần mềm theo mô hình phân rã mô-đun 127
5.2.5. Kiến trúc phần mềm theo miền ứng dụng 128
5.3. Một số phương pháp tạo kiến trúc 129
5.3.1. Phân hoạch ngang 129
5.3.2. Phân hoạch dọc 130
5.3.3. Phương pháp tạo kiến trúc từ biểu đồ luồng dữ liệu 130
5.4.Tiến trình thiết kế kiến trúc hệ thống phần mềm 131
5.4.1. Những hoạt động chung cho mọi tiến trình thiết kế 131
5.4.2. Quy trình thiết kế kiến trúc chương trình 132
5.5. Phân tích đặc tả yêu cầu phần mềm thành đặc tả kiến trúc phần mềm 134
5.5.1. Khái niệm luồng chuyển đổi và luồng giao dịch 135
5.5.2. Phân tích chuyển đổi thành kiến trúc phần mềm 137
5.5.3. Phân tích giao dịch thành kiến trúc phần mềm 140
5.6. Chất lượng thiết kế kiến trúc phần mềm 142
5.6.1. Một số tiêu chuẩn thiết kế kiến trúc phần mềm 142
5.6.2. Các hướng dẫn thiết kế kiến trúc 146
TÓM TẮT CHƯƠNG 147
CÂU HỎI CHƯƠNG 5 149
BÀI TẬP CHƯƠNG 5 150
Chương 6. ĐẶC TẢ KIẾN TRÚC MỘT SỐ HỆ THỐNG ĐƯỢC THIẾT KẾ THEO HƯỚNG HIỆN ĐẠI 151
6.1. Đặc tả kiến trúc hệ thống được thiết kế theo hướng phân tán 151
6.1.1. Sơ lược về hệ thống phân tán 151

6.1.2. Tổ chức hệ thống theo mạng địa phương 153
6.1.3. Tổ chức hệ thống theo kiến trúc Client-Server 154
6.1.4. Các đặc trưng của máy dịch vụ file và kiến trúc Client-Server 157
6.1.5. Các thành phần của mô hình Client-Server 158
6.1.6. Những dạng tiên tiến phổ dụng của kiến trúc Client-Server 158
6.1.7. Sự cân bằng giữa các yếu tố trong hệ phân tán 162
6.2. Đặc tả kiến trúc hệ thống được thiết kế theo hướng đối tượng 163
6.2.1. Mô hình hóa kiến trúc hệ thống 163
6.2.2. Phân tích kiến trúc 166
6.2.3. Thiết kế kiến trúc 172
6.2.4. Kiến trúc đối tượng phân tán 180
6.3. Đặc tả kiến trúc hệ thống được thiết kế theo hướng dịch vụ 191
6.3.1. Kiểu kiến trúc hệ thống được thiết kế theo hướng dịch vụ 191
6.3.2. Mô hình kiến trúc hướng dịch vụ 192
6.3.3. Các nguyên tắc thiết kế kiến trúc hướng dịch vụ 196
TÓM TẮT CHƯƠNG 199
CÂU HỎI CHƯƠNG 6 201
BÀI TẬP CHƯƠNG 6 201
Chương 7. BẢO ĐẢM CHẤT LƯỢNG PHẦN MỀM 202
7.1. Khái niệm bảo đảm chất lượng 202
7.2. Mô hình chất lượng 203
7.2.1. Các chuẩn đánh giá chất lượng sản phẩm phần mềm 203
7.2.2. Mô hình chất lượng trong và chất lượng ngoài 208
7.2.3. Mô hình chất lượng về mặt sử dụng 208
7.2.4. Mô hình phân cấp đặc tính 209
7.2.5. Mô hình tiến trình phần mềm 209
7.3. Việc đo chất lượng 210
7.3.1. Các chỉ tiêu đánh giá chất lượng phần mềm 210

7.3.2. Các chỉ tiêu đánh giá cổ điển 211
7.3.3. Các chỉ tiêu đánh giá chất lượng phần mềm theo ISO 9126 211
7.3.4. Các chỉ tiêu đánh giá chất lượng phần mềm khác 214
7.3.5. Quá trình phòng sạch 214
7.4. Các hoạt động bảo đảm chất lượng 215
7.4.1. Thẩm định và xác minh 215
7.4.2. Hoạt động thực hiện kiểm thử 216
7.4.3. Quản lý cấu hình phần mềm 219
7.4.4. Các hoạt động khác bảo đảm chất lượng phần mềm 220
7.5. Tiến trình triển khai bảo đảm chất lượng 220
7.5.1. Khảo sát nhu cầu 221
7.5.2. Lập kế hoạch triển khai 221
7.5.3. Triển khai 223
TÓM TẮT CHƯƠNG 228
CÂU HỎI CHƯƠNG 7 229
BÀI TẬP CHƯƠNG 7 230
Chương 8. KIỂM THỬ PHẦN MỀM 232
8.1. Chiến lược kiểm thử 232
8.1.1. Khái niệm chiến lược kiểm thử 232
8.1.2. Mô hình chiến lược tổng thể 233
8.1.3. Một số chiến lược kiểm thử bổ trợ 234
8.2. Các mức kiểm thử 236
8.2.1. Kiểm thử đơn vị 237
8.2.2. Kiểm thử tích hợp 241
8.2.3. Kiểm thử hệ thống 247
8.2.4. Kiểm thử chấp nhận/hợp lệ 251
8.3. Phương pháp và kỹ thuật kiểm thử 253
8.3.1. Kiểm thử hộp trắng 255

8.3.2. Kiểm thử hộp đen 263
8.3.3. Kiểm thử ứng dụng web 267
8.4. Thiết kế các ca kiểm thử 269
8.5. Tiến trình kiểm thử 271
8.6. Các công cụ kiểm thử 272
TÓM TẮT CHƯƠNG 277
CÂU HỎI CHƯƠNG 8 278
BÀI TẬP CHƯƠNG 8 279
Chương 9. NÂNG CẤP PHẦN MỀM 282
9.1. Bảo trì phần mềm 282
9.1.1. Khái niệm về bảo trì phần mềm 282
9.1.2. Tính bảo trì được 284
9.1.3. Dự đoán bảo trì và dự đoán thay đổi 285
9.2. Tiến trình nâng cấp phần mềm 285
9.3. Tái kỹ nghệ hệ thống 288
9.3.1. Khái niệm tái kỹ nghệ hệ thống 288
9.3.2. Tiến trình tái kỹ nghệ phần mềm 294
TÓM TẮT CHƯƠNG 305
CÂU HỎI CHƯƠNG 9 307
Chương 10. KỸ NGHỆ PHẦN MỀM HƯỚNG DỊCH VỤ 308
10.1. Kiến trúc hướng dịch vụ 308
10.1.1. Hướng tiếp cận dịch vụ 308
10.1.2. Các thành phần của một kiến trúc hướng dịch vụ 311
10.1.3. Các tính chất của một hệ thống SOA 313
10.2. Công nghệ dịch vụ 319
10.2.1. Nhận dạng kiểu dịch vụ thích hợp 319
10.2.2. Thiết kế dịch vụ 321
10.2.3. Các hoạt động phát triển dịch vụ 324

10.2.4. Một số công nghệ tiêu biểu xây dựng kiến trúc hệ thống theo hướng dịch vụ 324
10.3. Phát triển phần mềm bằng kỹ nghệ hướng dịch vụ 331
10.3.1. Một số mô hình triển khai SOA 331
10.3.2. Các nguyên tắc phát triển ứng dụng theo hướng dịch vụ 334
10.3.3. Vòng đời phát triển của dịch vụ 336
10.3.4. Các pha cơ bản xây dựng hệ thống SOA 337
10.3.5. Thiết kế và thực hiện luồng công việc 339
10.3.6. Các mức độ thực hiện kiến trúc hướng dịch vụ 344
TÓM TẮT CHƯƠNG 346
CÂU HỎI CHƯƠNG 10 347
TÀI LIỆU THAM KHẢO 349

LỜI NÓI ĐẦU

Ngay từ năm 1980, trong tạp chí Business Week, dòng tiêu đề "*Phần mềm - lực điều khiển mới hay phần mềm đã bước vào một thời đại*" đã đánh dấu chủ đề đáng quan tâm của các tạp chí, báo hiệu cho một cách hiểu mới về tầm quan trọng của phần mềm máy tính và đã đem đến những cơ hội và thách thức mới.

Công nghệ phần mềm (Software Technology) là tập hợp các kỹ thuật để tạo ra phần mềm. Công nghệ phần mềm được xem như là cơ sở tin học. Tích hợp thêm vào công nghệ phần mềm các khía cạnh khác như là các tiến trình sản xuất phần mềm theo quy tắc, kỷ luật và có bài bản, quản lý và đào tạo con người,... tạo nên miền khoa học rộng hơn, đó là *kỹ nghệ phần mềm (Software Engineering)*. Kỹ nghệ phần mềm hay công trình học phần mềm không chỉ là việc tạo ra sản phẩm mà nó liên quan đến việc sản sinh ra sản phẩm một cách hiệu quả.

Đến nay, công nghệ phần mềm đã có sự tiến hóa nhanh chóng. Nhiều hướng phát triển đã nối tiếp nhau sinh ra và lan rộng, trong đó có thể kể ra như hướng cấu trúc, hướng đối tượng, hướng thành phần, hướng dịch vụ, điện toán đám mây. Vì vậy, thử thách đối với kỹ sư phần mềm trở nên ngày càng cao để tạo ra phần mềm chất lượng cao, với những hạn chế về nguồn lực và phải tuân thủ một lịch trình định trước.

Tri thức về công nghệ phần mềm hay rộng hơn là kỹ nghệ phần mềm là một hành trang không thể thiếu của người làm công nghệ thông tin vì tính thực tiễn và lý luận khoa học tuyệt vời của nó. Cách tiếp cận hướng tới thực hành của những nhà khoa học đi trước như Pressman, Sommerville,... đang trường tồn và lan rộng.

Giáo trình này được soạn thảo nhằm mục đích cung cấp cho học viên cao học ngành công nghệ thông tin những kiến thức nâng cao về công nghệ phần mềm, bổ sung những kiến thức chuyên sâu hơn trong các giáo trình cơ sở mức

đại học. Với những kiến thức mở rộng này, các học viên có thể nâng cao kiến thức cơ sở của mình, tăng kỹ năng thực hành làm phần mềm và ứng dụng công nghệ thông tin vào thực tiễn.

Giáo trình *Công nghệ phần mềm nâng cao* gồm 10 chương:

Chương 1 - Tổng quan về công nghệ phần mềm: trình bày về những nét chung nhất về công nghệ phần mềm.

Chương 2 - Tiến trình phát triển phần mềm: trình bày về các mô hình tiến trình phát triển phần mềm cũng như các hoạt động của tiến trình phát triển phần mềm.

Chương 3 - Đặc tả yêu cầu phần mềm và yêu cầu hệ thống: trình bày về các tiến trình xác định yêu cầu và các đặc tả yêu cầu hệ thống cũng như phần mềm.

Chương 4 - Đặc tả thiết kế: trình bày về công nghệ thiết kế các phương diện quan trọng như dữ liệu, cấu trúc chương trình, giao diện, mô-đun xử lý.

Chương 5 - Đặc tả thiết kế kiến trúc: trình bày về các mô hình kiến trúc ứng dụng, một số phương pháp tạo kiến trúc tiêu biểu, tiến trình thiết kế kiến trúc hệ thống phần mềm, cách phân tích đặc tả yêu cầu phần mềm thành đặc tả kiến trúc phần mềm.

Chương 6 - Đặc tả kiến trúc một số hệ thống được thiết kế theo hướng hiện đại: trình bày về đặc tả kiến trúc của một số hệ thống được thiết kế theo hướng hiện đại như hướng phân tán, hướng đối tượng, hướng dịch vụ.

Chương 7 - Bảo đảm chất lượng phần mềm: trình bày về các vấn đề quan trọng trong việc đo chất lượng phần mềm.

Chương 8 - Kiểm thử phần mềm: trình bày về các mô hình chiến lược kiểm thử, các mức kiểm thử, các phương pháp và kỹ thuật kiểm thử, các phương án thiết kế ca kiểm thử, tiến trình kiểm thử cũng như các công cụ kiểm thử.

Chương 9 - Nâng cấp phần mềm: trình bày về các tiến trình nâng cấp phần mềm và tái kỹ nghệ.

Chương 10 - Công nghệ phần mềm hướng dịch vụ: trình bày về hướng tiếp cận, các thành phần, các tính chất của hệ thống có kiến trúc hướng dịch vụ, công nghệ dịch vụ, quy trình phát triển phần mềm bằng công nghệ hướng dịch vụ.

Đối tượng thụ hưởng chính của giáo trình này là các học viên cao học ngành Công nghệ thông tin của Trường Đại học Công nghệ thông tin và Truyền thông (CNTT&TT) thuộc Đại học Thái Nguyên.

Giáo trình này có thể dùng làm tài liệu học tập và tham khảo cho học viên cao học các chuyên ngành khác của công nghệ thông tin, cho nghiên cứu sinh về công nghệ thông tin, cho các giáo viên, các nhà nghiên cứu và tất cả những ai muốn đi sâu vào lĩnh vực công nghệ phần mềm.

Nhóm tác giả xin chân thành cảm ơn lãnh đạo Trường Đại học CNTT&TT, Đại học Thái Nguyên đã tạo điều kiện thuận lợi nhất để hoàn thành giáo trình này. Nhóm tác giả cũng xin chân thành cảm ơn PGS.TSKH. Nguyễn Xuân Huy, PGS.TS. Đoàn Văn Ban, PGS.TS. Bùi Thế Hồng, PGS.TS. Đặng Thành Phu, TS. Dương Chính Cương và các bạn đồng nghiệp đã cho chúng tôi nhiều ý kiến quý báu để hoàn thiện giáo trình này.

Mặc dù đã có rất nhiều cố gắng tham khảo các tài liệu trong ngoài nước, đúc rút kinh nghiệm từ nhiều năm tham gia các dự án và giảng dạy về công nghệ thông tin, giáo trình được biên soạn một cách cẩn trọng, công phu và tâm huyết, sửa chữa nhiều lần nhưng vẫn khó tránh khỏi những thiếu sót do tính quá rộng và khó của vấn đề trình bày. Chúng tôi chân thành mong đợi những nhận xét, ý kiến của các bạn đồng nghiệp, học viên và độc giả để giáo trình được hoàn thiện hơn.

Thư góp ý xin gửi về Phòng Đào tạo, Trường Đại học CNTT&TT, Đại học Thái Nguyên hoặc trực tiếp cho các tác giả theo địa chỉ: phunglevan52@gmail.com

Xin trân trọng cảm ơn./.

CÁC TÁC GIẢ

Chương 1
TỔNG QUAN VỀ CÔNG NGHỆ PHẦN MỀM

Phần mềm đã trở thành phần tử chủ chốt trong tiến hóa của các hệ thống và sản phẩm dựa trên máy tính. Công nghệ phần mềm là một chuyên ngành khoa học nghiên cứu tìm ra cách tốt nhất để làm ra các phần mềm đáp ứng các bài toán xử lý tin từ đơn giản đến phức tạp sao cho chương trình nhận được phải đúng, đáng tin và hiệu quả lại dễ sử dụng.

Chương này giới thiệu những đặc trưng chung nhất về công nghệ phần mềm. Học viên cần nắm vững các khái niệm về phần mềm, công nghệ phần mềm, các phương pháp trong kỹ nghệ phần mềm, bàn thợ CASE, các tiêu chuẩn của một phần mềm tốt, những thách thức chính đối với công nghệ phần mềm, khái quát được một số hướng kỹ nghệ phần mềm điển hình.

1.1. Định nghĩa phần mềm

Đến nay, đã có nhiều định nghĩa về phần mềm, song định nghĩa sau đây cho chúng ta một khái niệm khá đầy đủ về các thành phần phần mềm.

Định nghĩa [9]:

Phần mềm bao gồm một tập hợp các thành phần:

- Các tài liệu phân tích thiết kế, tài liệu hướng dẫn cài đặt và sử dụng phần mềm, tài liệu bảo trì, nâng cấp;

- Cơ sở dữ liệu (CSDL) được cài đặt trong môi trường thích hợp (hệ quản trị dữ liệu);

- Các dòng lệnh (chương trình máy tính) khả thi trên máy tính, phù hợp với hệ quản trị dữ liệu đã cài đặt CSDL;

- Các tiện ích số hoá đi kèm hỗ trợ cho chương trình máy tính.

Trong xã hội hiện đại, phần mềm giữ vai trò rất quan trọng. Phần mềm (SoftWare -SW) bây giờ đã giữ vị trí then chốt hơn phần cứng (HardWare-HW): điều mấu chốt cho sự thành công của nhiều hệ thống dựa trên máy tính. Khả năng lưu trữ của phần cứng biểu thị cho tiềm năng tính toán của máy tính. Còn phần mềm thể hiện một cơ chế giúp chúng ta chế ngự và khai thác tiềm năng này. Hệ thống phần mềm hoạt động trên các máy tính trở thành điều kiện sống còn của các máy tính đó.

Phần mềm là nhân tố đánh giá sự khác biệt; điều này thể hiện ở chỗ:

- Tính đầy đủ và đúng thời hạn của thông tin do phần mềm cung cấp (và các CSDL liên quan) dẫn đến sự khác biệt giữa một công ty này với các đối thủ cạnh tranh.

- Thiết kế và "tính thân thiện" của sản phẩm phần mềm cũng làm khác biệt nó với các sản phẩm cạnh tranh có cùng chức năng tương tự khác. Sự thông minh và chức năng do phần mềm được nhúng trong đó đưa ra thường làm khác biệt 2 sản phẩm tiêu thụ hay công nghiệp tương tự nhau. Như vậy, chính phần mềm tạo ra sự khác biệt đó.

Vai trò của phần mềm không ngừng mở rộng như Norman và Levy [23] đã viết: "Ngày nay, hoạt động tính toán có mặt khắp mọi nơi đã sản sinh ra một thế hệ những người ứng dụng thông tin sử dụng web để cung cấp một mạng lưới thông tin bao phủ liên kết mọi nhà, mọi công sở đến các xa lộ".

1.2. Định nghĩa công nghệ phần mềm

Công nghệ phần mềm (Software technology) là tập hợp các công nghệ (bao gồm nhiều kỹ thuật) để tạo ra phần mềm, mỗi công nghệ có thể có nhiều kỹ thuật tạo chương trình khác nhau [9]. *Chẳng hạn như Công nghệ hướng sự vật: một giải pháp để thiết kế và lập trình phần mềm hiệu quả nhờ sử dụng các sự vật được xây dựng từ trước. Công nghệ Java: một giải pháp tạo ra phần mềm có thể hoạt động trên các nền tảng khác nhau trên mạng máy tính.*

Có thể nói rằng, công nghệ phần mềm là tập hợp các phương pháp, quy trình, kỹ năng, bí quyết, công cụ, phương tiện để làm ra sản phẩm là phần mềm.

1.3. Sự khác biệt giữa công nghệ phần mềm với kỹ nghệ phần mềm

Trong cuộc hội thảo chính đầu tiên về chủ đề kỹ nghệ phần mềm, Fritz Bauer đã đưa ra định nghĩa ban đầu về kỹ nghệ phần mềm.

Định nghĩa:

Theo Naur P. [9,23], Kỹ nghệ phần mềm (Software Engineering) là việc thiết lập và sử dụng các nguyên lý công nghệ đúng đắn để thu được phần mềm một cách kinh tế vừa tin cậy vừa làm việc hiệu quả trên các máy thực.

Nhiều định nghĩa khác về sau đều nhấn mạnh vào các yêu cầu về một kỷ luật công nghệ trong việc phát triển. Parnas [23] coi kỹ nghệ phần mềm là việc xây dựng phần mềm nhiều phiên bản bởi nhiều người. Ghezzi [23] xem kỹ nghệ phần mềm là một lĩnh vực của khoa học máy tính, liên quan đến xây dựng các hệ thống phần mềm vừa lớn vừa phức tạp bởi một hay một số nhóm kỹ sư. Sommerville [25] xác định kỹ nghệ phần mềm là lĩnh vực liên quan đến lý thuyết, phương pháp và công cụ dùng cho phát triển phần mềm. Pressman [23] định nghĩa kỹ nghệ phần mềm là bộ môn tích hợp cả tiến trình, các phương pháp, các công cụ để phát triển phần mềm máy tính.

Trong công nghiệp phần mềm (Software Industry) *thì* công nghệ và kỹ nghệ không khác nhau là mấy, vì kỹ nghệ phần mềm không dùng đến các kỹ thuật gì khác hơn là kỹ thuật của công nghệ phần mềm. *Cái khác là cách nhìn: cách nhìn kỹ nghệ bao trùm hơn. Khía cạnh công nghệ thường là cách thức kỹ thuật để tạo ra những sản phẩm cụ thể (có thể xem như đó là định nghĩa cơ sở tin học), còn các khía cạnh khác của kỹ nghệ phần mềm, cho đến bây giờ thường dùng trong "việc huấn luyện trong công việc" là chủ yếu, nói tới việc thực hành các tiến trình sản xuất phần mềm theo quy tắc, kỷ luật và có bài bản. Nói gọn lại, kỹ nghệ phần mềm hay công trình học phần mềm không phải là chính việc sản sinh ra sản phẩm mà nó liên quan đến việc* sản sinh ra sản phẩm một cách hiệu quả.

1.4. Sự khác biệt giữa kỹ nghệ phần mềm và kỹ nghệ hệ thống

Vì hệ thống không phải là khái niệm nguyên thuỷ, nghĩa là chúng ta có thể dùng những khái niệm đã biết để định nghĩa thế nào là hệ thống, nhưng hệ thống là khái niệm được nghiên cứu và ứng dụng trong rất nhiều lĩnh vực khác

nhau nên việc nêu lên một định nghĩa hình thức hoá về hệ thống, có đủ tính khái quát để có thể áp dụng trong những lĩnh vực khác nhau là một việc vô cùng khó khăn. Tuy vậy, chúng ta sẽ đưa ra một "định nghĩa" phổ cập nhất.

Định nghĩa [13]:

Hệ thống bao gồm:

+ Tập hợp các phần tử (không phân biệt bản chất của nó).

+ Tập hợp các mối quan hệ giữa các phần tử đó (các quan hệ có nhiều dạng rất khác nhau. Có thể kể ra một vài dạng như các quan hệ cơ học, năng lượng, thông tin và các quan hệ khác ràng buộc bởi: kinh tế, thân hữu, pháp luật và có thể xác định qua các mặt như: số lượng, chiều hướng và cường độ của chúng).

+ Tạo thành một thể thống nhất để có được những chức năng hay mục tiêu (của chính nó hay được con người gán cho) của hệ thống.

Nhờ tạo thành một thể thống nhất mà hệ thống có được các đặc tính mà từng phần tử riêng rẽ không thể có được. Các đặc tính này gọi là các "đặc tính trồi".

Người kỹ sư hệ thống cần có sự hiểu biết về kỹ nghệ một hệ thống dựa trên máy tính vì những vấn đề của kỹ nghệ phần mềm thường là hậu quả tất yếu từ các quyết định kỹ nghệ ở mức toàn hệ thống.

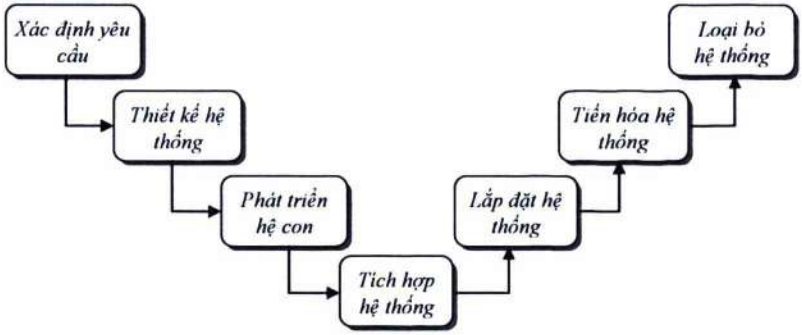
Kỹ nghệ hệ thống là hoạt động đặc tả, thiết kế, thực thi, thẩm định, triển khai và bảo trì toàn bộ hệ thống.

Định nghĩa về hệ thống tin học: Một hệ thống dựa trên máy tính gồm có các thiết bị tin học (phần cứng), các chương trình (phần mềm), các dữ liệu, con người và các thủ tục làm việc được tổ chức lại để tạo ra các chức năng hay dịch vụ phục vụ cho một tổ chức *[9,16]*.

Sự hoạt động thành công của mỗi thành phần trong một hệ thống phụ thuộc vào sự hoạt động của một số thành phần khác. Hệ thống luôn mang tính phân cấp *theo nghĩa là chúng luôn bao hàm trong nó những hệ thống nhỏ hơn (gọi là các hệ con) các thành phần của chúng. Khi các thành phần liên kết lại cấu thành một hệ thống, chúng tạo ra những* tính trồi *mà mỗi thành phần riêng rẽ không thể có được.*

Do phần mềm chỉ là một thành phần (cốt lõi) của hệ thống mà chúng ta quan tâm, nên khái niệm kỹ nghệ hệ thống bao hàm nhiều nghĩa và nội dung rộng hơn khái niệm kỹ nghệ phần mềm.

Tiến trình kỹ nghệ hệ thống tin học được thể hiện thông qua sơ đồ đặc tả trong hình 1.1.



Hình 1.1. Tiến trình kỹ nghệ thống dựa trên máy tính [16]

1.5. Các phương pháp trong kỹ nghệ phần mềm

Phương pháp cho một "cách làm" về mặt kỹ thuật để xây dựng phần mềm. Các phương pháp thường bao gồm phương pháp luận và các phương pháp cụ thể [16].

Phương pháp luận mô tả một cách tiếp cận mang tính kỹ thuật để tạo ra phần mềm, thường bao gồm các hoạt động theo một định hướng công nghệ và trình tự thực hiện chúng. Các hướng công nghệ như hướng cấu trúc, hướng dữ liệu, hướng đối tượng,... Các hoạt động bao gồm: lập kế hoạch và ước lượng dự án, phân tích yêu cầu hệ thống và phần mềm, thiết kế cấu trúc dữ liệu, kiến trúc chương trình và thuật toán, mã hoá, kiểm thử và bảo trì, và tùy thuộc hướng công nghệ được lựa chọn mà các phương pháp cụ thể thích hợp được sử dụng.

Các phương pháp cụ thể là cách thức giải quyết các vấn đề cụ thể phát sinh khi thực hiện các nhiệm vụ khác nhau trong các giai đoạn khác nhau của tiến trình phát triển phần mềm.

Tất cả các phương pháp đều dựa trên ý tưởng phát triển các mô hình của hệ thống mà có thể biểu diễn ở dạng đồ họa và dùng các mô hình này làm đặc tả hay thiết kế cho hệ thống được phát triển hay thẩm định. Các phương pháp thường bao gồm các thành phần sau (bảng 1.1):

Các khái niệm mô tả các phần tử của mô hình;

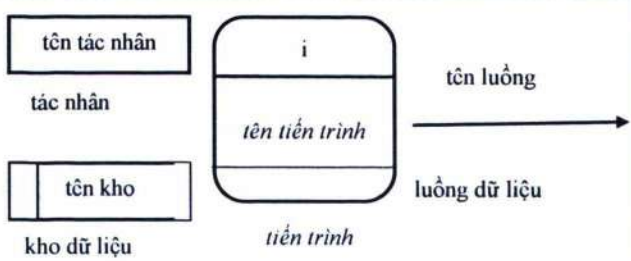
Các ký pháp (toán học, đồ họa) biểu diễn các phần tử mô hình;

Các quy tắc liên kết các phần tử tạo thành mô hình một cách đúng đắn;

Một quy trình hướng dẫn các bước để phát triển mô hình;

Những gợi ý, đề xuất cho việc vận dụng phương pháp vào những trường hợp thích hợp và để có được mô hình mô tả thế giới thực tốt nhất có thể.

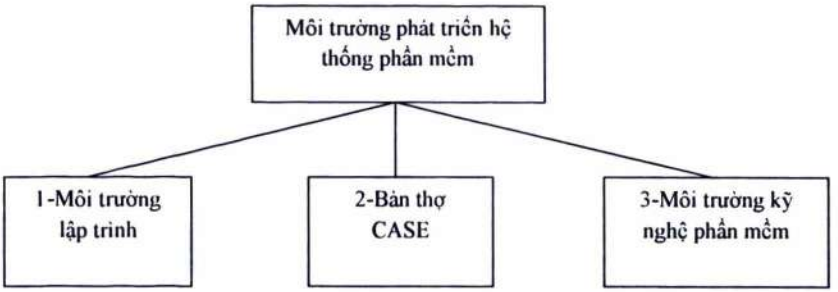
Bảng 1.1. Ví dụ về một phương pháp [16]

Thành phần của phương pháp	Phương pháp - Mô hình luồng dữ liệu
Các khái niệm	- Tác nhân: người, tổ chức, bộ phận tổ chức, hệ khác. - Tiến trình: dãy các hoạt động diễn ra tác động lên dữ liệu vào để tạo ra dữ liệu ra. - Luồng dữ liệu: các dữ liệu di chuyển từ một nơi đến nơi khác - Kho dữ liệu: các dữ liệu được lưu trữ tại một vị trí.
Ký pháp	

Các quy tắc	- Một tiến trình là duy nhất trong mô hình. - Các luồng dữ liệu vào một tiến trình phải khác các luồng dữ liệu ra. - Các luồng dữ liệu vào một tiến trình phải đủ để tạo ra các luồng dữ liệu ra. - Không có luồng dữ liệu vòng (quay lại nơi nó đi ra) - Không có các luồng dữ liệu giữa: kho dữ liệu - kho dữ liệu, kho dữ liệu - tác nhân, tác nhân - tác nhân
Các bước xây dựng	- Vẽ biểu đồ ngữ cảnh: chỉ có tác nhân, một tiến trình duy nhất mô tả cả hệ thống, các luồng dữ liệu giữa tác nhân và hệ thống (mô tả sự tương tác giữa môi trường với hệ thống) - Phát triển biểu đồ luồng dữ liệu mức đỉnh: xuất phát từ biểu đồ ngữ cảnh và các yếu tố của mô hình nghiệp vụ: + Thay tiến trình duy nhất bằng các tiến trình con + Giữ nguyên: các tác nhân ngoài và các luồng dữ liệu. Đặt đầu mút các luồng dữ liệu vào các tiến trình con tương ứng + Thêm: các kho dữ liệu, các luồng dữ liệu giữa kho-tiến trình, tiến trình - tiến trình - Phát triển các biểu đồ luồng dữ liệu mức i (i ≥1)...
Hướng dẫn	- Dùng biểu đồ luồng dữ liệu để biểu diễn mô hình ngữ cảnh của hệ thống. - Dùng biểu đồ luồng dữ liệu mô tả tiến trình xử lý nghiệp vụ ở các mức chi tiết dần: trước hết ở dạng vật lý (mô tả thế giới thực), sau ở dạng logic để phân tích.

1.6. Bàn thợ CASE

Môi trường phát triển hệ thống phần mềm bao gồm toàn bộ các công cụ phần cứng và phần mềm được sử dụng một cách thích hợp để trợ giúp việc tạo ra hệ thống phần mềm trong một lĩnh vực ứng dụng cụ thể. Có nhiều loại môi trường phát triển phần mềm khác nhau đang được sử dụng. Môi trường phát triển hệ thống phần mềm có thể phân thành 3 nhóm lớn [25] được mô tả trong hình 1.2.



Hình 1.2. Phân loại môi trường phát triển phần mềm

Môi trường lập trình là môi trường hướng vào trợ giúp lập trình, kiểm thử và gỡ rối. Sự trợ giúp của nó cho việc xác định yêu cầu, đặc tả và thiết kế phần mềm là rất hạn chế.

Bàn thợ CASE (Computer Aided Software Engineering) *là môi trường đặc biệt hướng vào trợ giúp cho đặc tả và thiết kế phần mềm. Chúng thường cung cấp rất ít hỗ trợ cho lập trình. Chúng có thể làm việc trên máy cá nhân và được sử dụng kết hợp với môi trường lập trình.*

Môi trường kỹ nghệ phần mềm là môi trường hướng vào việc trợ giúp để tạo ra các hệ thống phần mềm lớn, có cuộc sống dài mà chi phí bảo trì nó thường vượt quá chi phí phát triển và chúng thường do các đội phát triển chứ không phải do các nhà lập trình riêng rẽ thực hiện. Môi trường này thường hỗ trợ cho mọi hoạt động phát triển và bảo trì.

1.6.1. Khái niệm về CASE

Cho đến nay, không phải tất cả các hoạt động phát triển các hệ thống thông tin (HTTT) *đã được tự động hóa bằng máy tính mà không cần đến sự can thiệp của con người. Vì vậy, có rất nhiều công cụ đã được phát triển để trợ giúp cho các bước của tiến trình phát triển phần mềm mà ta gọi chung là* CASE, *tiếng Việt được dịch là* "Kỹ nghệ phần mềm được hỗ trợ bởi máy tính".

CASE có thể sử dụng để trợ giúp một phạm vi rộng rãi các hoạt động phát triển HTTT. Tuy nhiên, không một sản phẩm CASE đơn lẻ nào có thể cung cấp được các dịch vụ cho mọi bước của vòng đời hệ thống. Người ta thường phải dùng nhiều công cụ CASE để tạo nên những công cụ thích hợp. Ngày nay,

tính năng của chúng đã được tăng cường và đã làm cho tiến trình phát triển hệ thống ngày càng mang nhiều hơn các đặc tính kỹ thuật, tính nghệ thuật giảm đi và chất lượng của hệ thống đã tăng lên đáng kể.

Có nhiều công cụ khác nhau được xây dựng. Có công cụ thì đơn giản, có những công cụ lại rất phức tạp, mức độ được sử dụng chúng cũng không đồng đều. Jame Martin [23] đã liệt kê một loạt các công cụ được phát triển sau :

- Biểu đồ phân rã chức năng
- Biểu đồ luồng dữ liệu nghiệp vụ
- Biểu đồ luồng dữ liệu hệ thống
- Biểu đồ thực thể - mối quan hệ
- Biểu đồ phụ thuộc
- Biểu đồ hành động
- Biểu đồ phân tích dữ liệu
- Biểu đồ cấu trúc dữ liệu
- Biểu đồ di chuyển dữ liệu
- Cây quyết định và bảng quyết định
- Biểu đồ chuyển trạng thái
- Biểu đồ thiết kế đối thoại.
- Biểu đồ HIPOC (Hierarch Input Process Output Chart) [3]

trong đó 4 loại biểu đồ đầu tiên có tần suất sử dụng cao hơn cả.

1.6.2. Các loại công cụ CASE

Có một số cách phân loại CASE như sau:

- CASE phân loại theo các pha phát triển của vòng đời hệ thống;
- CASE phân loại theo tính năng;
- CASE phân loại theo quy mô.

1.6.2.1. CASE phân loại theo các pha phát triển của vòng đời hệ thống

Theo các pha phát triển của vòng đời hệ thống, CASE được phân thành hai loại:

1-Upper CASE: gồm các công cụ trợ giúp cho các nhà phát triển trong các pha đầu của tiến trình phát triển như xác định yêu cầu, phân tích và thiết kế.

2-Lower CASE: gồm các công cụ trợ giúp cho các nhà phát triển trong các pha sau của tiến trình phát triển như triển khai, tích hợp, vận hành và bảo trì.

1.6.2.2. CASE phân loại theo tính năng

Theo tính năng, CASE được phân thành 6 loại:

1- Công cụ lập biểu đồ: dùng để biểu diễn đồ thị các tiến trình của hệ thống, các dữ liệu và kiểm soát cấu trúc;

2- Công cụ tạo sinh báo cáo và màn hình: làm mẫu xem hệ thống được người sử dụng (NSD) *nhìn nhận và cảm thấy như thế nào. Chúng cũng giúp cho các nhà phân tích dễ dàng xác định yêu cầu dữ liệu và mối quan hệ của nó;*

3- Công cụ phân tích: dùng để kiểm tra một cách tự động, tính không đầy đủ, không nhất quán hoặc không chính xác của các đặc tả trong các biểu đồ, các biểu bảo và báo cáo;

4- Kho dữ liệu trung tâm (từ điển dữ liệu CASE): cho phép lưu trữ một cách tích hợp các đặc tả, các biểu đồ, các báo cáo và các thông tin quản lý dự án. Nó là công cụ tự động dùng để quản lý và quản lý truy nhập tới các thông tin của tổ chức và các thành phần ứng dụng trong kho dữ liệu. Từ điển chứa một danh sách được máy tính hóa gồm tất cả các dữ liệu được định nghĩa trong một sản phẩm. Trong sản phẩm lớn có đến hàng trăm các mục từ loại thì máy tính là công cụ lý tưởng để lưu trữ các thông tin này. Sức mạnh của từ điển dữ liệu là nó hỗ trợ để tổ hợp các dữ liệu theo các mô hình và kiểm tra sự nhất quán của chúng trong quá trình hình thành và phát triển các mô hình, làm các tài liệu đặc tả và tài liệu sử dụng, tạo sinh báo cáo, tạo sinh màn hình;

5- Bộ sinh tài liệu: giúp tạo ra các tài liệu kỹ thuật cũng như các tài liệu người dùng ở dạng chuẩn;

6- Bộ tạo sinh mã (chương trình): giúp việc tự động hóa việc sinh chương trình và cấu trúc dữ liệu trực tiếp từ tài liệu thiết kế, các biểu đồ, các biểu báo và báo cáo.

1.6.2.3. CASE phân loại theo quy mô

Các công cụ CASE được chia làm 3 mức tùy theo quy mô trợ giúp của nó cho tiến trình phát triển HTTT. CASE cũng có thể là những công cụ trợ giúp cho từng công đoạn riêng rẽ của tiến trình đó. CASE cũng có thể là bộ công cụ giúp tự động hóa một số các hoạt động mà chúng có liên hệ chặt chẽ với nhau để hoàn thành một nhiệm vụ. Ví dụ, một bàn thợ để đặc tả thiết kế, bàn thợ cho tổ chức khai thác dữ liệu,... Ở mức mạnh nhất, có bộ công cụ CASE (tools set)

là bộ được kết hợp với phần cứng hỗ trợ toàn bộ tiến trình phát triển phần mềm hay ít nhất cho phần chủ yếu của tiến trình đó.

Trong thời gian gần đây, bộ công cụ Designer/2000 và Developer/2000 được nhiều người biết đến và được sử dụng trong nhiều năm qua vì nó đã giúp trợ giúp được nhiều chức năng như lập biểu đồ, lập mô hình, kiểm tra chéo và tạo sinh các ứng dụng, tạo các mẫu biểu, tạo các báo cáo,...

1.6.3. Môi trường CASE tích hợp

Gene Forte [23] đã nói rõ "Trong khi các công cụ CASE riêng lẻ có đóng góp,... thì hứa hẹn về CASE thực sự nằm ở tiềm năng tích hợp nhiều công cụ vào một môi trường được tích hợp".

Lợi ích của CASE tích hợp (Integrated CASE, viết tắt là I-CASE) có thể kể ra như:

- Việc chuyển nhịp nhàng thông tin (mô hình, chương trình, dữ liệu) từ một công cụ này sang một công cụ khác và chuyển từ bước kỹ nghệ phần mềm này sang bước khác;
- Giảm bớt nỗ lực cần có để thực hiện các hoạt động như quản lý cấu hình phần mềm, bảo đảm chất lượng, sản xuất tài liệu;
- Làm tăng việc quản lý dự án thông qua việc lập kế hoạch, điều phối và truyền thông tốt hơn;
- Sự phối hợp được nâng cao giữa các thành viên cùng làm việc trong một dự án phần mềm lớn.

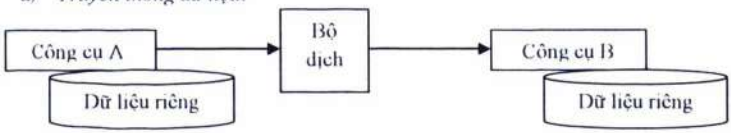
Tuy vậy, I-CASE cũng đặt ra các thách thức. Việc tích hợp đòi hỏi cách biểu diễn nhất quán về thông tin kỹ nghệ phần mềm, các giao diện được chuẩn hóa giữa các công cụ, một cơ chế thuần nhất cho trao đổi giữa kỹ sư phần mềm và mỗi công cụ, một cách tiếp cận hiệu quả sẽ làm cho I-CASE đi vào trong số nhiều nền phần cứng và hệ điều hành khác nhau.

Có nhiều cách tích hợp các công cụ CASE. Trong thực tế, một vài công cụ CASE được dùng hoàn toàn độc lập. Các tùy chọn tích hợp sau đây là sẵn có:

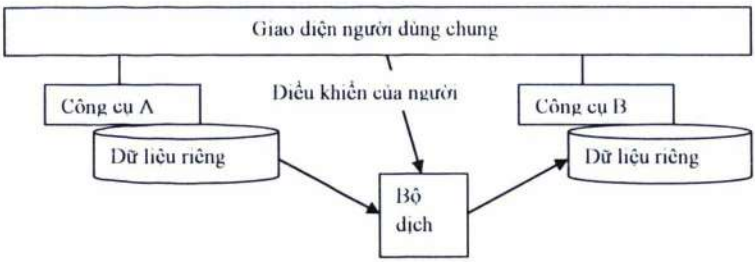
- Truyền thông dữ liệu;
- Sử dụng công cụ chung;
- Tích hợp dữ liệu;
- Tích hợp đầy đủ.

Hình 1.3 thể hiện các mức của công việc tích hợp CASE.

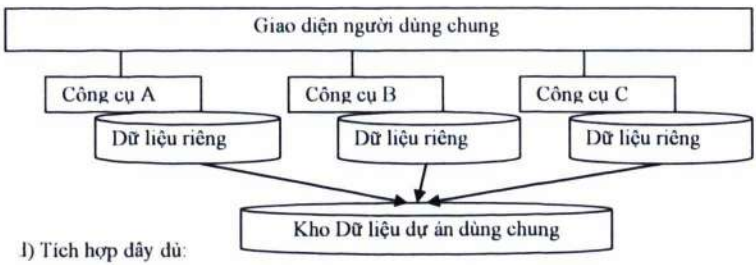
a) Truyền thông dữ liệu:



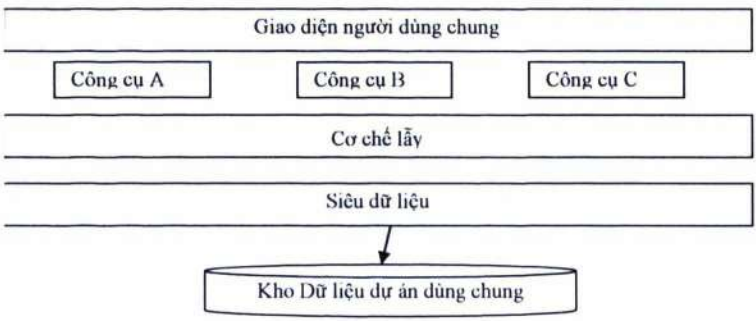
b) Sử dụng công cụ



c) Quản lý dữ liệu chung:



I) Tích hợp đầy đủ:



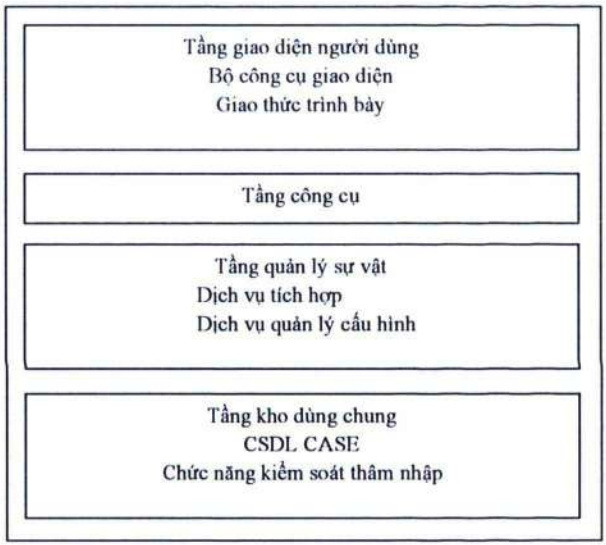
Hình 1.3. Các mức của công việc tích hợp CASE [23]

Khi dùng CASE, một vốn chung các thông tin kỹ nghệ phần mềm sẽ được tạo ra. Khung tích hợp tạo điều kiện thuận lợi cho việc truyền thông tin vào và ra vốn chung này. Để thực hiện được điều này, các thành phần kiến trúc sau đây phải tồn tại:

- Một CSDL được tạo ra để quản lý thông tin

- Một hệ thống quản lý sự vật được xây dựng để quản lý việc thay đổi thông tin.

- Một cơ chế quản lý công cụ được xây dựng để điều hòa việc sử dụng công cụ CASE, một giao diện người dùng phải có sẵn để cung cấp đường đi nhất quán giữa những hành động do người sử dụng (NSD) tiến hành và những công cụ có trong môi trường. Phần lớn các mô hình về khung tích hợp đều biểu diễn những thành tố này theo các tầng. Một mô hình đơn giản về khung, chỉ mô tả những thành tố đã được lưu ý ở trên được vẽ trong hình 1.4.



Hình 1.4. Mô hình kiến trúc cho khung tích hợp

Khi một môi trường CASE tích hợp được xem xét tới thì cơ chế để tích hợp các công cụ CASE sẽ được cài đặt khác nhau tùy thuộc vào kiến trúc, nền và triết lý của người thiết kế môi trường. Tuy nhiên, mọi môi trường đều cài

đặt cơ chế thực hiện và cơ chế truyền thông. Người ta thường dùng mô hình môi trường công cụ của Cộng đồng Châu Âu phát triển - là một trong một số chuẩn về môi trường CASE tích hợp. Trong đó, các cơ chế thực hiện và truyền thông được coi là cơ chế cơ sở. Đó là các chức năng được xác định để thao tác các "thực thể" vốn tồn tại trong hoàn cảnh phát triển phần mềm. Các thực thể bao gồm cả các sự vật (như dữ liệu, chương trình gốc, tài liệu, thiết bị) và các công cụ thao tác trên các sự vật này.

1.7. Tiêu chuẩn của một phần mềm tốt

Việc đánh giá tổng quát về chất lượng hệ thống bao gồm việc đánh giá chất lượng tài liệu đặc tả, chất lượng thiết kế và chất lượng của phần mềm công trình (kỹ nghệ) tốt thông qua các thuộc tính chung.

Bốn thuộc tính chủ chốt mà một hệ phần mềm tốt hẳn là phải có [9]:

1- Có thể bảo trì được: phần mềm tuổi thọ dài phải được viết và được lập tư liệu sao cho việc thay đổi có thể tiến hành được mà không quá tốn kém

2- Đáng tin cậy: phần mềm phải thực hiện được điều mà người tiêu dùng mong mỏi và không thất bại nhiều hơn những điều đã được đặc tả.

3- Có hiệu quả: hệ thống phải không lãng phí nguồn lực bộ nhớ, bộ xử lý. Không đòi hỏi phải cực đại hoá độ hiệu quả vì rằng việc đó có thể làm cho phần mềm rất khó thay đổi.

4- Có giao diện người sử dụng thích hợp: giao diện người sử dụng phải phù hợp với khả năng và kiến thức của người dùng hệ thống.

Giá cả phải được tính đến khi xây dựng một phần mềm công trình tốt.

Bảo trì được coi là thuộc tính chủ chốt vì rằng các chi phí gắn kết với sản phẩm phần mềm chủ yếu là trong giai đoạn phần mềm đó được đưa vào sử dụng.

Việc tối ưu hoá mọi thuộc tính trên là rất khó khăn. Quan hệ giữa chi phí và sự cải thiện từng thuộc tính không phải là tuyến tính và các cải thiện nho nhỏ trong bất kỳ thuộc tính nào cũng là rất đắt.

Năm 1991, chuẩn ISO/IEC 9126 được đưa ra nhằm quy định các đặc tính về chất lượng phần mềm. Có 6 đặc tính lớn được quan tâm trong chuẩn này [9]:

1- Tính chức năng (Functionality)

Tính chức năng liên quan đến khả năng thực hiện một loạt các thao tác để đạt được các yêu cầu hiện hay ẩn của NSD.

2- Tính tin cậy (Reability)

Tính tin cậy là đặc tính liên quan đến khả năng bảo toàn hiệu quả của phần mềm dưới những tác động của các điều kiện hay ở mức giới hạn nào đó.

3- Tính sử dụng được (Usability)

Tính sử dụng được là đặc tính liên quan khả năng sử dụng phần mềm theo hướng giảm thiểu những nỗ lực của người sử dụng.

4- Tính hiệu quả (Efficiency).

Tính hiệu quả gồm bao gồm cả tính hiệu quả thời gian và tính hiệu quả tài nguyên).

5- Tính bảo trì được (Maintenability).

Tính bảo trì được thể hiện thông qua tính phân tích được, tính thay đổi được, tính ổn định và tính kiểm tra được.

6- Tính khả chuyển (Portability).

Tính khả chuyển được thể hiện thông qua tính thích nghi được, tính cài đặt được, tính tuân thủ, tính có thể thay thế được.

1.8. Những thách thức chính đối với công nghệ phần mềm

Trong những năm gần đây, các thách thức đối với phát triển phần mềm máy tính gia tăng vì những nguyên nhân sau [16]:

- Sự tinh vi và năng lực của phần cứng đã vượt xa khả năng xây dựng phần mềm để có thể sử dụng được các tiềm năng của nó.

- Khả năng xây dựng các phần mềm mới không giữ được cùng nhịp so với nhu cầu về phần mềm tăng lên nhanh chóng, đặc biệt khi Internet phát triển.

- Quy mô và độ phức tạp của các phần mềm mới ngày càng tăng. Khả năng bảo trì các hệ thống phần mềm cũ hiện đang tồn tại rất khó khăn và tốn kém các nguồn tài nguyên vì các thiết kế sơ sài. Phát triển các phần mềm mới phải nhanh chóng và dễ bảo trì trở thành nhu cầu cấp bách.

Tất cả các thách thức trên đã dẫn đến việc chấp nhận thực hành kỹ nghệ phần mềm để có thể tạo nhanh các phần mềm với chất lượng ngày một cao, có quy mô và số lượng ngày một lớn và có những tính năng tương ứng với tiềm năng phần cứng.

1.9. Tổng quan một số hướng kỹ nghệ phần mềm điển hình

Kể từ năm 1970 đến nay, kỹ nghệ phần mềm đã có sự tiến hóa nhanh chóng. Nhiều hướng kỹ nghệ đã nối tiếp nhau sinh ra và phát triển, trong đó có thể kể ra như kỹ nghệ hướng cấu trúc, kỹ nghệ hướng đối tượng, kỹ nghệ hướng thành phần, kỹ nghệ hướng dịch vụ, điện toán đám mây. Trong mục này, chúng ta sẽ khái quát các đặc trưng chính của chúng.

1.9.1. Kỹ nghệ hướng cấu trúc

Tiếp cận hướng cấu trúc là bước tiếp tục phát triển của hướng dữ liệu. Người ta thường gộp hai cách tiếp cận dữ liệu và chức năng làm một và gọi là tiếp cận hướng cấu trúc. Tiếp cận hướng cấu trúc hướng vào việc cải tiến cấu trúc các chương trình dựa trên cơ sở mô-đun hoá các chương trình để dễ theo dõi, quản lý, bảo trì. Tiếp cận hướng cấu trúc thể hiện ra bằng tiến trình phát triển hệ thống hướng cấu trúc.

Phân tích có cấu trúc là một hoạt động xây dựng mô hình, trong đó có sử dụng một ký pháp duy nhất và tạo ra các mô hình mô tả cho luồng và nội dung thông tin, phân hoạch hệ thống theo chức năng và hành vi và mô tả bản chất của điều phải xây dựng. Đây là một phương pháp kỹ nghệ phần mềm đã thịnh vượng và thu được việc sử dụng rộng rãi trong toàn bộ cộng đồng kỹ nghệ phần mềm [23].

Các phương pháp luận hướng cấu trúc sử dụng một số công cụ để xác định luồng thông tin và các tiến trình xử lý. Việc xác định và chi tiết hoá dần dần các luồng dữ liệu và các tiến trình là ý tưởng cơ bản phương pháp luận

theo phương pháp top-down. Vì vậy, Pressman đã nói rằng: "Kỹ nghệ hướng cấu trúc bao gồm các kỹ thuật mô hình hóa luồng và nội dung thông tin".

Một hệ thống tin học hóa được biểu diễn như một phép biến đổi thông tin duy nhất (một số cái vào thành một số cái ra). Một biểu đồ luồng dữ liệu là một kỹ thuật vẽ ra luồng thông tin và những biến đổi được áp dụng khi dữ liệu đi từ đầu vào đến đầu ra. Biểu đồ luồng dữ liệu có thể được dùng để biểu diễn cho một hệ thống hay phần mềm ở bất kỳ mức trừu tượng nào. Trong thực tế, nó được biểu diễn ở nhiều mức, cao nhất là mức ngữ cảnh, dưới đó là mức đỉnh, dưới nữa là các mức dưới đỉnh các cấp. Pressman gọi mô hình luồng dữ liệu với mức đỉnh là "mức 0" hay là "mô hình hệ thống nền tảng".

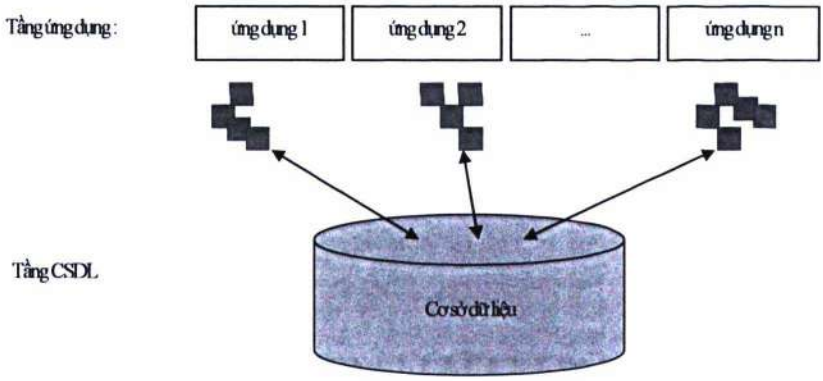
Biểu đồ luồng dữ liệu là một công cụ đồ họa có thể rất có giá trị trong phân tích yêu cầu phần mềm.

Đặc tính cấu trúc của một HTTT hướng cấu trúc được thể hiện trên 3 cấu trúc chính [10]:

1- Cấu trúc dữ liệu (mô hình quan hệ);

2- Cấu trúc hệ thống chương trình (cấu trúc phân cấp điều khiển các mô-đun và phần chung);

3- Cấu trúc chương trình và mô-đun (cấu trúc một chương trình và ba cấu trúc lập trình cơ bản).



Hình 1.5.Cấu trúc hệ thống hướng cấu trúc

Kỹ nghệ hướng cấu trúc đã cung cấp một tập hợp đầy đủ các đặc tả hệ thống ít dư thừa được phát triển theo tiến trình logic và lặp lại. Nó cho ta nhiều lợi ích sau:

- Làm giảm sự phức tạp (nhờ phương pháp top-down);

- Tập trung vào ý tưởng (trước hết tập trung vào mô hình ý tưởng của HTTT, bỏ qua những ràng buộc vật lý. Sau khi được mô hình ý tưởng mới bổ sung và cụ thể hoá gắn vào các phương tiện vật lý;

- Chuẩn hoá (các định nghĩa, công cụ và cách tiếp cận chuẩn mức cho phép nhà thiết kế làm việc tách biệt, đồng thời với hệ thống con khác nhau mà không cần có liên kết với nhau vẫn đảm bảo sự thống nhất trong dự án;

- Hướng về tương lai (tập trung vào đặc tả một hệ thống đầy đủ, hoàn thiện và mô-đun hoá cho phép thay đổi (bảo trì) dễ dàng khi hệ thống đi vào hoạt động;

- Giảm bớt tính nghệ thuật trong thiết kế (các nhà thiết kế buộc phải tuân thủ các quy tắc và nguyên tắc phát triển đối với nhiệm vụ phát triển, giảm sự ngẫu hứng quá đáng.

So sánh với nhiều cách tiếp cận khác, cách tiếp cận hướng cấu trúc có nhiều đặc điểm nổi trội [10].

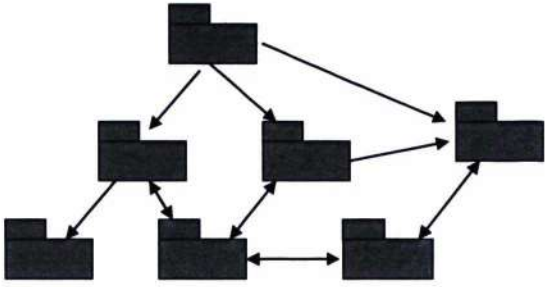
1.9.2. Kỹ nghệ hướng đối tượng

Cách tiếp cận mới hơn hướng cấu trúc để phát triển hệ thống là cách tiếp cận hướng đối tượng (Object-Oriented Approach-OOA) [2,11]. Hệ thống được xây dựng chỉ gồm các thành phần liên kết với nhau được gọi là đối tượng. Các đối tượng thường tương ứng với các vật thực trong HTTT. Chẳng hạn như khách hàng, nhà cung cấp, hợp đồng. Mục tiêu của kỹ nghệ này là làm cho các phần tử hệ thống trở nên độc lập tương đối với nhau và có thể dùng lại. Điều đó đã cải thiện cơ bản chất lượng của hệ thống và làm tăng năng suất của hoạt động phát triển hệ thống. Theo cách tiếp cận này, hệ thống được coi là một bộ các đối tượng. Đối tượng là bộ các "thuộc tính" xác định trạng thái của đối tượng đó và các phép toán thực hiện trên các thuộc tính đó.

Ý tưởng khác nằm ở phía sau của cách tiếp cận này là sự kế thừa và bao gói thông tin. Các đối tượng được tổ chức thành từng lớp là một nhóm các đối tượng có cùng cấu trúc và hành vi. Kế thừa cho phép tạo ra lớp mới mà có chung với các lớp đang tồn tại một số đặc trưng mới. Chẳng hạn, từ lớp đối tượng "người", ta có thể sử dụng kế thừa để xác định một lớp khác là "khách hàng". Cả hai lớp này đều có chung đặc trưng như tên, địa chỉ, số điện thoại,... Nhưng lớp "người" là tổng quát hơn lớp "khách hàng". Như vậy, mỗi đối tượng được thừa kế *từ một vài lớp đối tượng lớp cao hơn. Các đối tượng liên lạc với nhau chỉ bằng cách trao đổi các thông báo (liên lạc giữa các đối tượng thực hiện bằng cách một đối tượng này gọi một thủ tục).*

Cách tiếp cận hướng đối tượng dựa trên ý tưởng che giấu thông tin. *Che giấu thông tin là chiến lược thiết kế giấu càng nhiều thông tin trong các thành phần càng hay. Liên lạc thông qua các thông tin trạng thái dùng chung (các biến tổng thể) là ít nhất, nhờ vậy khả năng hiểu được nâng lên.*

Do bao gói cả dữ liệu và xử lý trong một đối tượng mà hoạt động của nó không ảnh hưởng đến các đối tượng khác. Các đối tượng được ghép nối với nhau thành một hệ thống thông qua mối quan hệ truyền thông bằng cách gửi và nhận thông điệp. Rõ ràng rằng, với cơ chế này, hệ thống được "lắp ghép" và "tháo gỡ" đơn giản, dễ bảo trì, dễ sử dụng lại. Cách tiếp cận mới này đáp ứng được những nhu cầu và thách thức cơ bản hiện nay trong phát triển phần mềm: Cần phát triển những hệ thống phần mềm có quy mô lớn, phức tạp hơn, nhanh, dễ bảo trì và chi phí chấp nhận được.



Hình 1.6. Cấu trúc hệ thống hướng đối tượng

Hiện nay đã có những ngôn ngữ như UML (Unified Modeling Language) và ngôn ngữ lập trình hướng đối tượng (như C++, Eiffel, Object PASCAL, Java, Ada...) có các cơ chế trợ giúp phát triển các đối tượng, kế thừa và đóng gói.

Kỹ thuật phát triển hệ thống hướng đối tượng đã phát triển và đang lan rộng. Về tổng thể, nhiệm vụ của phân tích hướng đối tượng là xác định các đối tượng, xác định cấu trúc, hành vi của nó, xác định mối quan hệ giữa chúng. Nhiệm vụ đầu tiên của thiết kế hướng đối tượng là mô hình hoá các chi tiết về hành vi của đối tượng và truyền thông với các đối tượng khác sao cho những yêu cầu của hệ thống được đáp ứng, kiểm tra lại và xác định lại các đối tượng để tận dụng ưu thế tính kế thừa và các lợi ích khác của hướng đối tượng.

Mô hình hướng đối tượng là rất thích hợp cho việc phân tích và thiết kế phần mềm lớn và phức tạp [5,11].

Tuy nhiên chúng ta cũng cần nhận rõ các hạn chế của cách tiếp cận hướng đối tượng như:

- Sự tường minh các đối tượng hệ thống thích hợp là khó khăn. Cách nhìn tự nhiên nhiều hệ thống là cách nhìn chức năng và việc thích nghi với cách nhìn hướng đối tượng đôi khi là khó khăn, nhất là đối tượng trừu tượng, không phải đối tượng cụ thể.

- Phương pháp thiết kế hướng đối tượng tuy đã phát triển và đang thay đổi nhanh chóng, song phương pháp hiện đang được sử dụng rộng rãi nhất vẫn là phương pháp dựa trên sự phân rã chức năng (hướng cấu trúc).

1.9.3. Kỹ nghệ hướng thành phần

Trong thời kỳ sau 1990, một trong những đặc điểm nổi trội là hướng tới phát triển phần mềm dựa trên công nghệ khoa học và tin cậy hơn như là kỹ nghệ phần mềm hướng thành phần. Có nhiều mô hình thành phần tiêu biểu trên thế giới như là COM, DCOM của Microsoft, CORBA của OMG, JavaBeans, Enterprise JavaBeans của Sun và .Net Assemblies [9].

Hình dung một modem có thể được chế tạo bằng cách lắp ráp từ các thành phần IC khác nhau, tương tự như vậy, các hệ thống phần mềm có thể được xây dựng bằng việc viết cạnh nhau các thành phần phần mềm khác nhau.

Các thành phần cho phép sản phẩm hoá phần mềm. Do vậy, các thành phần phần mềm đã trở nên sản phẩm thương mại có giá rẻ.

Hiện nay, người ta đã thừa nhận một khái niệm chung về một thành phần như sau [9]: Thành phần là một bộ phận chức năng phân tán, độc lập, có khả năng cung cấp quyền truy cập tới các dịch vụ của nó thông qua giao diện. *Trong định nghĩa này, nhấn mạnh một số khía cạnh quan trọng của một thành phần:*

- Trước tiên, nó xác định một thành phần như một đơn vị phân tán. Do đó, nó có đặc trưng của một gói thực thi phần mềm.

- Thứ hai, nó chỉ ra một thành phần cung cấp một số chức năng hữu ích được tập hợp cùng nhau để thỏa mãn một số nhu cầu. Nó được thiết kế để cung cấp chức năng dựa trên một số chuẩn thiết kế.

- Thứ ba, một thành phần cung cấp các dịch vụ qua các giao diện. Để sử dụng thành phần, cần phải thực hiện qua giao diện, chứ không phải bằng cách truy cập các chi tiết thực thi bên trong của thành phần.

Người ta nói rõ hơn về những đặc trưng nổi trội của một thành phần [9]: Một thành phần là độc lập. Nói rõ hơn, một thành phần là một thực thể thực thi độc lập có thể thực hiện một hay nhiều đối tượng thực thi.

Các thành phần cung cấp một dịch vụ mà không quan tâm tới nơi thành phần đang thực thi hoặc ngôn ngữ lập trình của nó.

Một thành phần cung cấp chức năng qua giao diện. Mỗi thành phần có thể có nhiều giao diện khác nhau, mỗi giao diện biểu diễn một dịch vụ mà thành phần cung cấp. Giao diện thành phần được công khai và tất cả các tương tác đều thông qua các giao diện đó.

Có thể tổ hợp/kết hợp các thành phần với nhau thành một thành phần phức hợp. Đây là đặc trưng quan trọng nhất. Để thành phần có thể kết hợp được với nhau, chúng phải có các đặc tả rõ ràng cái gì chúng yêu cầu và cái gì chúng cung cấp. Chúng phải chỉ rõ cái gì môi trường triển khai sẽ cần để cung cấp cho việc thực thi thành phần, ví dụ, các giao diện yêu cầu của các thành phần khác. Ngoài ra, chúng phải chỉ rõ một danh sách các dịch vụ cung cấp tất cả các điều kiện lỗi có thể xảy ra.

là một thành phần không có trạng thái liên tục. Đặc trưng này của thành phần thường không thể hiện rõ. Nó không thể được phân biệt với các bản sao của nó. Một thành phần có thể được kích hoạt và được tải vào một hệ thống. Các thể hiện khác nhau của thành phần có thể có sẵn trong thời gian chạy.

Có 5 phần tử chính trong một thành phần. Mỗi phần tử của một thành phần cũng là một đặc trưng trong cách tiếp cận dựa trên thành phần [9]:

1- Phần đặc tả;

2- Phần thực thi;

3- Một chuẩn;

4- Các gói;

5- Phần triển khai.

Phát triển hệ thống dựa trên thành phần có thể được chia thành 4 pha:

1- Xác định thành phần - xác định thành phần đang tồn tại là thích hợp cho việc sử dụng trong hệ thống mới, giữ nguyên các yêu cầu chức năng và phi chức năng.

2- Thích nghi thành phần - điều chỉnh thành phần để thực hiện các yêu cầu của hệ thống mới. Đo mức độ phù hợp của thành phần phụ thuộc vào các thành phần riêng lẻ

3- Lắp ráp các thành phần - tích hợp các thành phần để tạo thành một hệ thống phần mềm từ các thành phần khác nhau

4- Tiến hóa hệ thống - thêm mới, xóa, hoặc thay thế một thành phần để nâng cấp hệ thống, thêm các chức năng mới.

Lợi ích của công nghệ dựa trên thành phần bao gồm khả năng sử dụng lại, khả năng hiểu được, giảm chi phí phát triển và tính mở rộng động. Tuy nhiên, đặc trưng quan trọng nhất để tính toán thay đổi là khả năng thích nghi. Các hệ thống thành phần có khả năng thay đổi các cấu hình bằng cách thêm mới, thay thế và xóa bỏ các thành phần cấu tạo, làm thích nghi phần mềm như yêu cầu.

Kỹ nghệ phần mềm dựa trên thành phần (Component - Based Software Engineering - CBSE) *là một tiến hoá tự nhiên của tiến trình kỹ nghệ phần*

mềm. Những điểm nổi trội trong tiến trình phát triển phần mềm dựa trên thành phần gồm có:

- CBSE là một cách tiếp cận phát triển phần mềm dựa trên việc sử dụng lại phần mềm.

- CBSE xuất hiện để khắc phục lỗi theo cách phát triển hướng đối tượng đến việc hỗ trợ sử dụng lại hiệu quả, các lớp đối tượng đơn là quá chi tiết và cụ thể.

- Các thành phần trừu tượng hơn các lớp đối tượng và có thể được xem là các bộ cung cấp dịch vụ độc lập.

Trong quá trình phát triển, các mô hình thành phần nổi bật có thể liệt kê như sau:

- Mô hình đối tượng thành phần (Component Object Model-COM) là một kiến trúc cũ về phần mềm hướng thành phần được đề xuất đầu tiên bởi Microsoft năm 1995. COM+ là phiên bản mới nhất của mô hình COM của Microsoft. Thực thi mô hình thành phần COM+ là nền tảng cho nhiều kỹ thuật mức cao hơn, bao gồm IIS, ASP, Site Server, Application Center 2000, và Biztalk Server 2000.

- Enterprise JavaBeans (EJB) là một mô hình thành phần phần mềm dành cho việc phát triển và triển khai các ứng dụng nghiệp vụ mức doanh nghiệp. Mô hình bao gồm một đặc tả mở (Matena and Hapner, 1999) mà người sản xuất thành phần, người tiêu dùng, NSD cuối và người sản xuất mô hình thành phần phần mềm đều có thể sử dụng để phát triển các sản phẩm phần mềm của họ.

- Mô hình hoạt động kinh doanh Java Bean được Sun Microsystems phát triển, là một mô hình thành phần để xây dựng phần mềm hướng thành phần về mặt dịch vụ. Mô hình Java bean là thành phần dùng lại cơ sở trong EJB.

- Do các hạn chế với mô hình đối tượng CORBA trước đó, OMG thông qua mô hình thành phần CORBA (CORBA Component Model- CCM) để mở rộng và gộp mô hình đối tượng CORBA. CCM gồm các đặc tả trong CORBA 3.0, được phát hành năm 2001. Phạm vi ứng dụng của CORBA ngày càng rộng

nên CCM được đặt vào vị trí phù hợp để sử dụng trong các ứng dụng Client-Server then chốt.

1.9.4. Kỹ nghệ hướng dịch vụ

Kỹ nghệ hướng dịch vụ mà kiến trúc hướng dịch vụ (Service Oriented Architecture-SOA) *là hướng phát triển mới nhất, đang được xem như sự phát triển kế tiếp của các hướng kỹ nghệ trên, được kỳ vọng tạo nên một cuộc cách mạng mới trong việc phát triển phần mềm. Trong kỹ nghệ phần mềm hướng dịch vụ, việc tách riêng phần thực hiện dịch vụ (mô-đun phần mềm) với giao tiếp gọi dịch vụ tạo nên một cách thức nhất quán cho các ứng dụng khai thác sử dụng dịch vụ mà không cần quan tâm tới công nghệ thực hiện dịch vụ.*

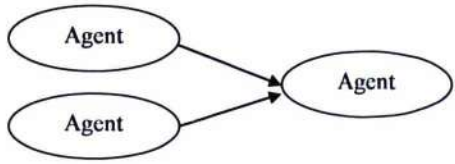
Tại diễn đàn kiến trúc hướng dịch vụ do tạp chí InfoWorld tổ chức năm 2006, SOA được nhận định là một khung làm việc lớn, trong đó các doanh nghiệp xây dựng, triển khai và quản lý các dịch vụ; Các dịch vụ này là các thành phần ứng dụng; Chúng có thể được các ứng dụng khác gọi ra sử dụng thông qua các giao thức chuẩn.

Ở mức độ chung nhất, SOA được hiểu là một loại kiến trúc về hệ thống trong đó mỗi ứng dụng được xem như một nguồn cung cấp dịch vụ. Nói rõ hơn, SOA là một kiến trúc phần mềm dựa trên một tập các dịch vụ, mỗi dịch vụ đảm nhận một chức năng cơ bản dưới dạng mô-đun phần mềm có nhiệm vụ thực hiện một qui trình nghiệp vụ nào đó. Các dịch vụ trong SOA được xác định rõ ràng, có thể thay đổi, cập nhật, nâng cấp, tích hợp mới. Các dịch vụ có thể giao tiếp với nhau, độc lập với phần nền của hệ thống và có thể tái sử dụng. Các giao tiếp có thể chỉ là việc chuyển giao dữ liệu đơn giản hoặc là gồm ít nhất hai dịch vụ hợp tác thực hiện một hoạt động cần thiết nào đó.

Theo Gartner, 2003, SOA là hướng tiếp cận thiết kế phần mềm kiểu Client-Server, trong đó một ứng dụng gồm có các dịch vụ phần mềm và các nhà tiêu dùng các dịch vụ phần mềm (cũng có thể hiểu như khách hàng hoặc người yêu cầu dịch vụ). SOA khác mô hình Client-Server ở chỗ xác định nhấn mạnh vào việc lắp ghép lỏng lẻo giữa các cấu phần phần mềm và việc sử dụng các giao diện tách biệt nhau.

Trong SOA, có hai cách trao đổi thông tin giữa các đại lý-Agent (đại diện của các ứng dụng) [14]:

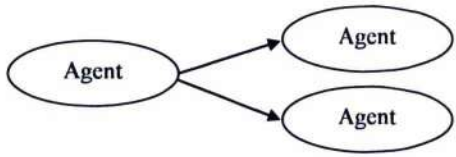
1- Điểm tới điểm (Point-to-Point)



Hình 1.7. Point-to-Point

Cách (mẫu hình) trao đổi thông tin này được sử dụng khi cần gửi thông báo từ nhiều ứng dụng cho một ứng dụng cụ thể. Điều này xuất hiện trong các kiến trúc hướng dịch vụ khi nhiều đại lý cần gửi tin nhắn cho hàng đợi tin tức và sau đó chúng được chuyển đến cho một điểm đích.

2- Xuất tin - Đăng ký (Publish - Subscribe)



Hình 1.8. Publish - Subscribe

Cách trao đổi thông tin này được sử dụng khi đăng ký cho nhiều ứng dụng cùng nhận một tin nhắn. Điều này xuất hiện trong các kiến trúc hướng dịch vụ, khi một đại lý gửi một thông điệp đến hàng đợi và tin nhắn này sẽ được chuyển cho tất cả các điểm đích đã được thẩm định [14].

Nói đến SOA là nói đến "tiết kiệm", cả tiết kiệm chi phí lẫn thu được giá trị nhiều hơn từ các hệ thống có sẵn. Có nhiều lý do để hàng trăm tập đoàn đã chú ý đến SOA như một giải pháp tích hợp nhằm giảm giá thành của một đề án công nghệ thông tin (CNTT) một cách rất ấn tượng [22] như sử dụng lại những thành phần có sẵn, đó là một giải pháp ứng dụng tổng hợp cho doanh nghiệp,

có tính kết nối lỏng giúp tăng tính linh hoạt và khả năng triển khai cài đặt, thích ứng với những thay đổi trong tương lai, hỗ trợ đa thiết bị và đa nền tảng, tăng khả năng mở rộng và khả năng sẵn sàng cung cấp,...

Kiến trúc hướng dịch vụ thật sự đem đến những lợi ích to lớn. Thế nhưng việc xây dựng, ứng dụng, quản lý nó như thế nào không phải là việc đơn giản.

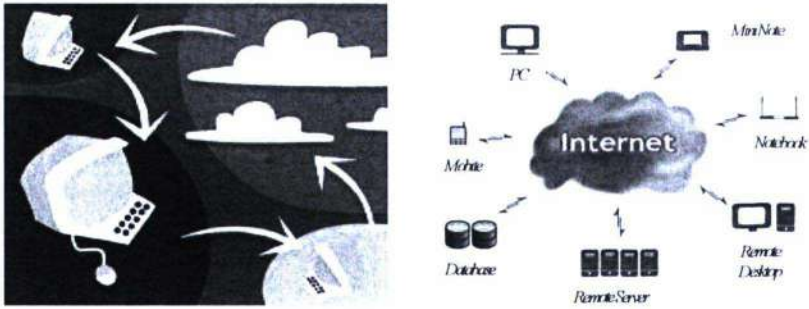
Trên thế giới, các công ty CNTT như IBM, Microsoft, BEA, HP, SAP, Tibco và Oracle là những nhà cung cấp giải pháp phần mềm kiến trúc hướng dịch vụ hàng đầu hiện nay.

Trong chương 6, chúng ta sẽ nghiên cứu sâu hơn về kiến trúc hệ thống được thiết kế theo hướng dịch vụ và trong chương 10, chúng ta sẽ nghiên cứu các vấn đề còn lại của kỹ nghệ hướng dịch vụ.

1.9.5. Điện toán đám mây

Điện toán đám mây (Cloud Computing*), còn gọi là* điện toán máy chủ ảo, *là mô hình* tính toán *sử dụng các công nghệ máy tính và phát triển dựa vào mạng* Internet [22].

Thuật ngữ "Điện toán đám mây" ra đời giữa năm 2007 không phải để thể hiện một trào lưu mới, mà để khái quát lại các hướng phát triển của cơ sở hạ tầng CNTT vốn đã và đang diễn ra từ những năm qua. Quan niệm này có thể được diễn giải một cách đơn giản: các nguồn tính toán khổng lồ như các phần cứng (máy chủ), phần mềm, và các dịch vụ (chương trình ứng dụng), ..., sẽ nằm tại các máy chủ ảo (đám mây) trên Internet thay vì trong máy tính gia đình và văn phòng (trên mặt đất) để mọi người kết nối và sử dụng mỗi khi họ cần. Nói cách khác, ở mô hình tính toán này, mọi khả năng liên quan đến CNTT đều được cung cấp dưới dạng các "dịch vụ", cho phép NSD truy cập các dịch vụ CNTT từ một nhà cung cấp nào đó "trong đám mây" mà không cần phải biết về công nghệ đó, cũng như không cần quan tâm đến các cơ sở hạ tầng phục vụ công nghệ đó.



Hình 1.9. Mô hình "Điện toán đám mây"[22]

Theo tổ chức IEEE (Institute Electrical and Electronic Engineers*): "Điện toán đám mây là hình mẫu trong đó thông tin được lưu trữ thường trực tại các máy chủ trên Internet và chỉ được lưu trữ tạm thời ở các máy khách, bao gồm máy tính cá nhân, trung tâm giải trí, máy tính trong doanh nghiệp, các phương tiện máy tính cầm tay, ...*" [26].

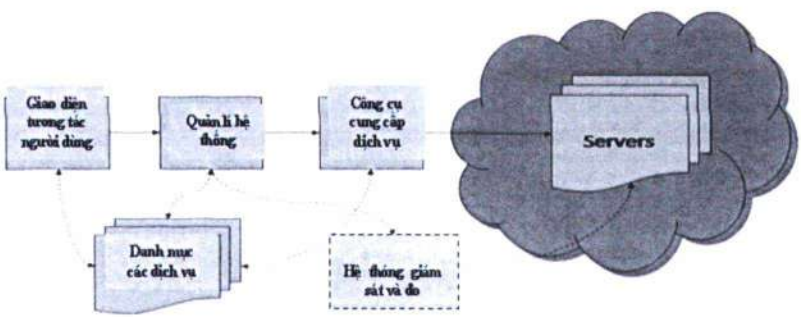
Điện toán đám mây là khái niệm tổng thể bao gồm cả các khái niệm như phần mềm dịch vụ, Web 2.0 và các vấn đề khác xuất hiện gần đây, các xu hướng công nghệ nổi bật, trong đó đề tài chủ yếu của nó là vấn đề dựa vào Internet để đáp ứng những nhu cầu tính toán của người dùng. Ví dụ, dịch vụ Google Apps cung cấp những ứng dụng kinh doanh trực tuyến thông thường, có thể truy nhập từ một trình duyệt web, còn các phần mềm và dữ liệu đều được lưu trữ trên các máy chủ.

"Điện toán đám mây" là sự kết hợp nhiều công nghệ hiện có như: Kiến trúc hướng dịch vụ, ảo hóa, điện toán tự trị với những ý tưởng mới để tạo ra một giải pháp CNTT đầy đủ. *Điện toán đám mây là một giải pháp bao gồm tất cả, trong đó các tài nguyên điện toán như phần cứng, phần mềm, mạng, lưu trữ,... được cung cấp nhanh chóng cho người dùng theo họ yêu cầu.*

Các nguồn tài nguyên hoặc các dịch vụ được phân phát có thể quản trị để bảo đảm khả năng sẵn sàng cao, an ninh và chất lượng. Yếu tố chính cho các giải pháp này là chúng sở hữu khả năng điều chỉnh tăng và giảm, để cho người dùng có được những tài nguyên mà họ cần. Các giải pháp "điện toán đám mây" cho phép CNTT được cung cấp như một dịch vụ. Đối tượng phục vụ chủ yếu hiện nay của CNTT là các tổ chức, các cơ sở doanh nghiệp.

Phần lớn hạ tầng cơ sở của điện toán đám mây hiện nay là sự kết hợp của những dịch vụ đáng tin cậy được phân phối thông qua các trung tâm dữ liệu (data center) được xây dựng trên những máy chủ với những cấp độ khác nhau của các công nghệ ảo hóa. Những dịch vụ này có thể được truy cập từ bất kỳ đâu trên thế giới, trong đó "đám mây" là một điểm truy cập duy nhất cho tất cả các máy tính có nhu cầu của khách hàng.

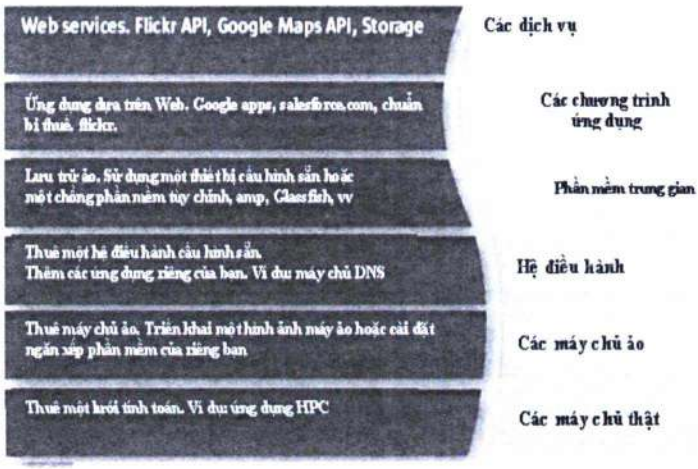
Các dịch vụ thương mại cần đáp ứng yêu cầu chất lượng dịch vụ từ phía khách hàng và thông thường đều đưa ra các mức thỏa thuận dịch vụ. Các tiêu chuẩn mở (Open standard) và phần mềm mã nguồn mở (Open source software) cũng góp phần phát triển điện toán đám mây.



Hình 1.10. Mô hình kiến trúc điện toán đám mây [15]

Tất cả các tài nguyên tính toán (phần cứng, phần mềm) được tổ chức thành danh mục các dịch vụ (Services Catalog). Các dịch vụ này được cung cấp lên mạng Internet ở các Server trong các đám mây thông qua các công cụ cung cấp dịch vụ (Provisioning Tool). Các dịch vụ được tương tác với NSD thông qua các phần giao diện tương tác NSD (User Interaction Interface). Tất cả các dịch vụ được quản lý bởi Quản lý hệ thống (Systems Management) và được theo dõi bởi hệ thống giám sát (Monitoring & Metering).

Theo kiến trúc do Sun đề xuất, điện toán đám mây cung cấp các dịch vụ ở tất cả các tầng, từ phần cứng tới các phần mềm như trong hình sau [14 Viện CNTT (2009]:



Hình 1.11. Cơ sở hạ tầng các tầng của điện toán đám mây [15]

TÓM TẮT CHƯƠNG

Công nghệ phần mềm là tập hợp các phương pháp, quy trình, kỹ năng, bí quyết, công cụ, phương tiện để làm ra sản phẩm là phần mềm.

Kỹ nghệ phần mềm là việc thiết lập và sử dụng các nguyên lý công nghệ đúng đắn để thu được phần mềm một cách kinh tế vừa tin cậy vừa làm việc hiệu quả trên các máy thực. Kỹ nghệ phần mềm không phải là chính việc sản sinh ra sản phẩm mà nó liên quan đến việc sản sinh ra sản phẩm một cách hiệu quả.

Giữa kỹ nghệ phần mềm với công nghệ phần mềm, kỹ nghệ hệ thống có sự khác biệt.

Phương pháp trong kỹ nghệ phần mềm cho một "cách làm" về mặt kỹ thuật để xây dựng phần mềm. Các phương pháp thường bao gồm phương pháp luận và các phương pháp cụ thể.

Bàn thợ CASE là môi trường đặc biệt hướng vào trợ giúp cho đặc tả và thiết kế phần mềm. Chúng thường cung cấp rất ít hỗ trợ cho lập trình. Chúng có thể làm việc trên máy cá nhân và được sử dụng kết hợp với môi trường lập trình.

Việc đánh giá tổng quát về chất lượng hệ thống bao gồm việc đánh giá chất lượng tài liệu đặc tả, chất lượng thiết kế và chất lượng của kỹ nghệ tốt thông qua các thuộc tính chung. Bảo trì được coi là thuộc tính chủ chốt vì rằng các chi phí gắn kết với sản phẩm phần mềm chủ yếu là trong giai đoạn phần mềm đó được đưa vào sử dụng.

Năm 1991, chuẩn ISO/IEC 9126 được đưa ra nhằm quy định các đặc tính về chất lượng phần mềm. Có 6 đặc tính lớn và được thể hiện thông qua 21 đặc tính con.

Các thách thức đối với phát triển phần mềm máy tính gia tăng đã dẫn đến việc chấp nhận thực hành kỹ nghệ phần mềm để có thể tạo nhanh các phần mềm với chất lượng ngày một cao, có quy mô và số lượng ngày một lớn và có những tính năng tương ứng với tiềm năng phần cứng.

Kể từ năm 1970 đến nay, kỹ nghệ phần mềm đã có sự tiến hóa nhanh chóng. Nhiều hướng kỹ nghệ đã nối tiếp nhau sinh ra và phát triển, trong đó có thể kể ra như kỹ nghệ hướng cấu trúc, kỹ nghệ hướng đối tượng, kỹ nghệ hướng thành phần, kỹ nghệ hướng dịch vụ, điện toán đám mây.

CÂU HỎI CHƯƠNG 1

1.1. Anh/Chị hãy thử đưa ra một định nghĩa phần mềm của riêng mình?

1.2. Công nghệ phần mềm khác và giống kỹ nghệ phần mềm, kỹ nghệ hệ thống ở điểm nào?

1.3. Phương pháp luận và các phương pháp cụ thể khác nhau ở chỗ nào?

1.4. CASE dùng để làm gì?

1.5. Anh/Chị hãy tự đưa ra một nhóm thuộc tính chủ chốt mà một hệ phần mềm tốt hẳn phải có?

1.6. So sánh với nhiều cách tiếp cận khác, cách tiếp cận hướng cấu trúc có những đặc điểm nổi trội nào?

1.7. Nêu rõ các ưu điểm và hạn chế của cách tiếp cận hướng đối tượng?

1.8. Tại sao người ta phải dùng nhiều phương pháp tiếp cận khác nhau (hướng cấu trúc hoặc hướng đối tượng) trong việc thiết kế các thành phần khác nhau của một hệ thống?

1.9. Chiến lược nào (hướng cấu trúc hoặc hướng đối tượng) tốt nhất cho các dự án lớn? Vì sao?

1.10. Các cách tiếp cận hướng cấu trúc và hướng đối tượng có đối kháng nhau?

BÀI TẬP CHƯƠNG 1

Bài tập 1.1:

Có một hệ thống kinh doanh bán buôn được mô tả như sau:

Một công ty kinh doanh bán buôn một số mặt hàng. Công ty tiếp nhận các Đơn hàng *của KHACH và sau đó kiểm tra tính hợp lệ của đơn hàng. Nếu Đơn hàng hợp lệ thì kiểm tra Khách là cũ hay mới. Nếu khách cũ có nợ quá hạn (*rà soát Sổ công nợ*) hay khách mới (*đối chiếu với Danh sách khách*) thì* chuyển đơn hàng *sang bộ phận giải quyết khách, ngược lại* chuyển Đơn hàng *cho bộ phận xử lý Đơn hàng.*

Tại bộ phận giải quyết khách, nếu là khách mới người ta sẽ trao đổi với khách nhằm hiểu rõ thêm về khách, sau đó thực hiện ký một hợp đồng *nếu thấy cần thiết; Còn đối với khách nợ quá hạn, người ta sẽ phải* giải quyết nợ cũ *và* quyết định *có cho phép khách tiếp tục mua hàng mới hay không.*

Tại bộ phận xử lý Đơn hàng, người ta đối chiếu Đơn hàng với Thẻ kho để biết được khả năng đáp ứng yêu cầu của khách. Trong trường hợp thiếu hàng hay có các yêu cầu mặt hàng mới, người ta sẽ thoả thuận với khách, sau đó lập Phiếu giao hàng và phiếu thu rồi chuyển chúng *cho bộ phận gom và gửi hàng.*

Bộ phận gom và gửi hàng sẽ tổ chức *gom hàng theo Phiếu giao hàng và gửi hàng cho khách. Trước khi gửi hàng, người ta thoả thuận với khách về phương thức gửi và* giao cho khách một Phiếu giao hàng kèm theo một phiếu thu.

Yêu cầu:

1-Vẽ biểu đồ phân rã chức năng cho hệ thống trên?

2-Vẽ biểu đồ luồng dữ liệu mức 0 (mức đỉnh) cho hệ thống?

Bài tập 1.2:

Một loại chứng từ được công ty kinh doanh bán buôn sử dụng có dạng:

ĐƠN ĐẶT HÀNG

Số:

Tên khách hàng: *Ngày đặt:*

Địa chỉ: *Điện thoại:*

Số TT	*Tên hàng*	*Mô tả hàng*	*Đơn vị*	*Số lượng*

Yêu cầu:

Vẽ biểu đồ thực thể - mối quan hệ dựa trên chứng từ ĐƠN ĐẶT HÀNG

Chương 2
TIẾN TRÌNH PHÁT TRIỂN PHẦN MỀM

Chất lượng phần mềm được xây dựng phụ thuộc rất lớn vào tiến trình phát triển phần mềm. Do vậy, tri thức về tiến trình phát triển phần mềm là rất hữu ích cho các công trình sư phần mềm.

Chương này trình bày về tiến trình phần mềm và các hoạt động cơ bản của nó.

Học viên cần nắm vững khái niệm về tiến trình và mô hình tiến trình, biết cách sử dụng các mô hình tiến trình như mô hình thác nước, mô hình tiến hóa. Học viên cần biết đầy đủ các hoạt động của tiến trình phát triển phần mềm cũng như các tài liệu cần đưa ra trong mỗi hoạt động.

2.1. Tiến trình phần mềm

2.1.1. Khái niệm tiến trình phần mềm

Khi ta xây dựng một phần mềm một cách công nghiệp, điều quan trọng là phải vạch ra được một loạt các bước hoạt động dự kiến trước - một lộ trình, mà theo đó ta có thể tạo ra một phần mềm chất lượng, hiệu quả và đúng thời hạn. Một lộ trình như vậy được gọi là một *tiến trình phần mềm (software process)* [16].

Định nghĩa trên cho thấy, đặc trưng trước hết của tiến trình là *các hoạt động* cần thực hiện. Trong các hoạt động đó phải có *các hoạt động mang tính kỹ thuật* để tạo ra sản phẩm. Mặt khác, sản phẩm phải bảo đảm các chất lượng và các ràng buộc đặt ra. Vì vậy, loại hoạt động cần thiết nữa là *các hoạt động quản lý.*

Đặc trưng thứ hai của tiến trình là *trình tự thực hiện các hoạt động*. Để tiện cho quản lý, các hoạt động thường được phân chia vào các bước. Trình tự thực hiện hoạt động trong tiến trình được thể hiện qua trình tự thực hiện các

bước. Tất nhiên các bước có thể được thực hiện tuần tự hoặc có thể lặp lại một số bước nào đó.

2.1.2. Khái niệm mô hình tiến trình

Mô hình tiến trình phần mềm là sự mô tả tiến trình một cách đơn giản hóa khi xem xét nó từ một cách nhìn cụ thể. Về bản chất, mô hình tiến trình là một sự trừu tượng hóa một lớp các tiến trình thực. Một vài mô hình tiến trình phần mềm được nhiều người biết đến như mô hình thác nước, mô hình xoắn ốc, mô hình làm bản mẫu, mô hình sử dụng ngôn ngữ thế hệ bốn.

Vào năm 1996, Huff [23] đã lần đầu tiên hệ thống hóa các mô hình tiến trình phần mềm. Sommerville [25] đưa ra một số loại mô hình tiêu biểu:

– Mô hình thác nước (waterfall model);

– Mô hình phát triển tiến hóa (evolutionary models)*: là mô hình phát triển với quá trình lặp để hoàn thiện dần phần mềm;*

– Phát triển hệ thống hình thức (formal system development)*: một cách tiếp cận dựa trên đặc tả hệ thống bằng toán học để chứng minh hay chuyển đổi nó thành chương trình nhờ các công cụ toán học chuyên dụng;*

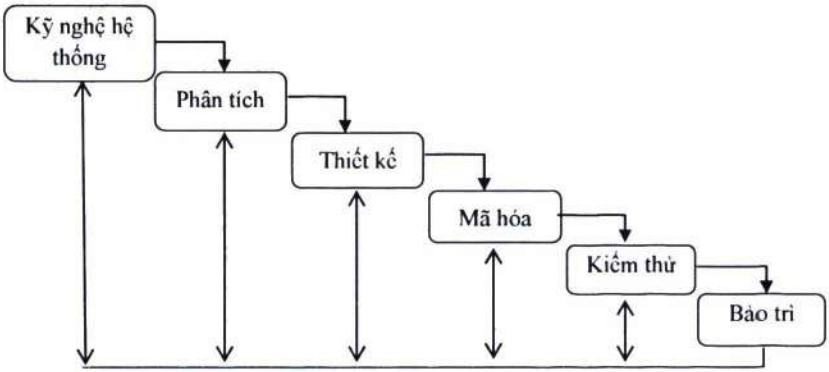
– Phát triển phần mềm theo hướng sử dụng lại (reuse oriented sofware development)*: quá trình phát triển tập trung vào việc tích hợp các thành phần đã có để nhận được hệ thống, đáp ứng được các yêu cầu đặt ra.*

2.2. Mô hình thác nước

Mô hình thác nước hay còn gọi là mô hình vòng đời cổ điển do Royce đề xuất năm 1970. Nó yêu cầu tiếp cận một cách hệ thống, tuần tự và chặt chẽ đối với việc phát triển phần mềm, bắt đầu ở mức hệ thống và tiến dần xuống phân tích, thiết kế, mã hóa, kiểm thử và bảo trì (hình 2.1).

Mô hình vòng đời cổ điển là mô hình có sớm nhất và được sử dụng rộng rãi nhất. Nó đưa ra các phương pháp khoa học, các bước tổng quát áp dụng được cho nhiều mô hình. Nó là tốt hơn nhiều lần cách tiếp cận tự nhiên. Tuy nhiên, các dự án phần mềm thực tế hiếm khi tuân theo trình tự tuần tự. Quá trình lặp lại là không thể tránh khỏi. Khi áp dụng mô hình này, nếu không quay lại thì dễ gặp bất trắc, mà lặp lại thì khó quản lý được tiến độ và không đáp ứng

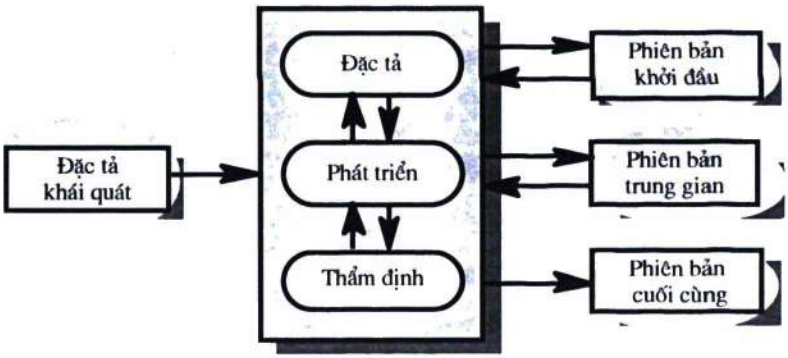
kịp thời yêu cầu của khách hàng. Mặt khác, khách hàng phải kiên nhẫn đến cuối dự án mới có chương trình làm việc được. Thời gian thực hiện dự án theo mô hình này thường dài, các tài liệu đặc tả rất lớn. Nếu chương trình gặp lỗi khi kết thúc mới phát hiện ra thì có thể sẽ là một thảm họa.



Hình 2.1 Mô hình thác nước của tiến trình phần mềm

2.3. Mô hình tiến hóa

Ý tưởng chung của phương pháp tiến hóa là phát triển một phiên bản sơ khai ban đầu và giới thiệu để người dùng bình luận rồi làm mịn nó thông qua các bước cho đến khi nhận một hệ thống thích hợp.



Hình 2.2. Mô hình phát triển tiến hóa [16]

Phương pháp này thích hợp khi mà chúng ta khó thực hiện các đặc tả chi tiết ngay từ đầu. Phương pháp thiếu đặc tả thường hay được dùng để phát triển các hệ thống trí tuệ nhân tạo nhằm kích thích những khả năng tư duy của con người. Chìa khóa thành công trong cách tiếp cận này là cho phép thực hiện các bước lặp hệ thống một cách nhanh chóng: những sự thay đổi như những gợi ý được phối hợp với việc trình diễn nhanh đến mức có thể. Nó yêu cầu những ngôn ngữ lập trình có trình độ rất cao để phát triển các phần mềm như LISP, Prolog. Biểu đồ tiến trình tổng quát của nó được đặc tả như hình 2.2. Với phương pháp này ta không có mô tả đầy đủ nên không thể kiểm tra và không thể nói đến một chương trình đúng.

2.3.1. Mô hình làm bản mẫu

Làm bản mẫu *(prototyping)* là tạo ra một mô hình như thực tế cho phần mềm cần xây dựng. Mô hình có thể có 3 dạng:

1-Bản mẫu trên giấy *hay trên máy tính cá nhân (PC) mô tả giao diện người - máy làm người dùng hiểu được cách các tương tác xuất hiện. Nó là một bản mô tả yêu cầu;*

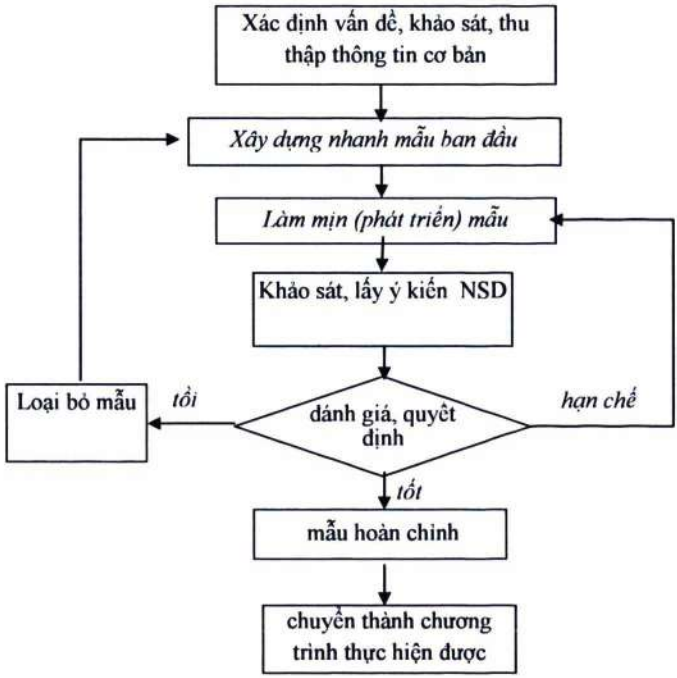
2-Bản mẫu cài đặt *chỉ chứa một tập con chức năng của phần mềm mong đợi;*

3-Bản mẫu là một chương trình có thể thực hiện *một phần hay tất cả chức năng mong muốn nhưng sơ lược cần cải tiến thêm các tính năng khác.*

Trước hết, người phát triển và khách hàng gặp nhau và xác định mục tiêu tổng thể cho phần mềm, xác định các yêu cầu đã biết, các miền cần khảo sát thêm. Rồi thiết kế nhanh bản mẫu ban đầu. Thiết kế nhanh tập trung vào việc biểu diễn các khía cạnh của phần mềm thấy được đối với người dùng (cách đưa vào và định dạng đưa ra) sau đó làm mịn mẫu, trình diễn để người dùng đánh giá rồi tiếp tục làm mịn các yêu cầu cho phần mềm. Tiến trình này lặp đi lặp lại cho đến khi bản mẫu thỏa mãn yêu cầu của khách, đồng thời giúp người phát triển hiểu kỹ hơn nhu cầu nào cần phải thực hiện (hình 2.3).

Cách tiếp cận làm bản mẫu cho kỹ nghệ phần mềm là cách tiếp cận thực tế nhất. Nó rất thích hợp với các hệ thống vừa và nhỏ, đặc biệt khi yêu cầu chưa rõ ràng, chưa xác định cụ thể cái vào cái ra và chưa chắc chắn nội dung

xử lý hay hiệu quả của một thuật toán. Nó được sử dụng hiệu quả khi kết hợp với các mô hình khác, chẳng hạn, ở giai đoạn xác định yêu cầu của mô hình thác nước,...



Hình 2.3 Mô hình làm bản mẫu

2.3.2. Mô hình xoắn ốc

Mô hình xoắn ốc, được Boehm đưa ra năm 1988, dựa trên ý tưởng là tối thiểu hóa rủi ro bằng việc phân tích yếu tố rủi ro ở mỗi bước lặp và sử dụng phương pháp làm bản mẫu. Quá trình phát triển được chia thành nhiều bước lặp lại, mỗi bước bắt đầu bằng việc lập kế hoạch, phân tích rủi ro rồi tạo bản mẫu, hoàn thiện và phát triển hệ thống, duyệt lại, và cứ thế tiếp tục (hình 2.4). Nội dung một bước gồm bốn hoạt động chính:

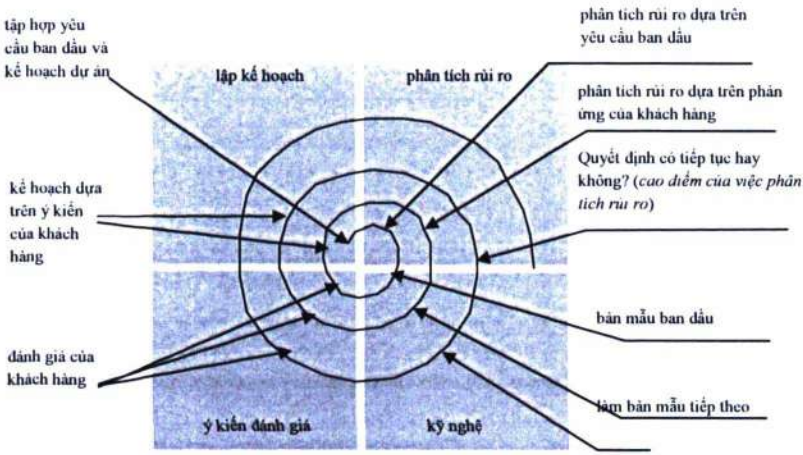
1- Lập kế hoạch*: xác định mục tiêu, giải pháp và ràng buộc;*

2- Phân tích rủi ro: *phân tích các phương án, xác định và giải quyết rủi ro;*

3- Kỹ nghệ: *phát triển sản phẩm "mức tiếp theo";*

4- Đánh giá của khách hàng: *khẳng định kết quả của kỹ nghệ.*

Với mỗi lần lặp xoắn ốc (bắt đầu từ tâm), các phiên bản được hoàn thiện dần. Nếu phân tích rủi ro chỉ ra rằng yêu cầu không chắc chắn thì bản mẫu có thể được sử dụng trong góc phần tư kỹ nghệ; các mô hình và các mô phỏng khác cũng được dùng để làm rõ hơn vấn đề và làm mịn yêu cầu. Tại một vòng xoắn ốc, phân tích rủi ro phải đi đến quyết định "tiến hành tiếp hay dừng". Nếu rủi ro quá lớn thì có thể đình chỉ dự án hay thay đổi yêu cầu đặt ra cho thích hợp.



Hình 2.4. Mô hình xoắn ốc [9]

Mô hình này thích hợp để phát triển các hệ thống quy mô lớn. Trong mô hình này, không có sự phân biệt rõ ràng giữa hoạt động bảo trì và phát triển. Tuy nhiên, việc thay đổi một cách linh hoạt khi phát triển đòi hỏi nhà phát triển và khách hàng phải có liên kết chặt chẽ, nếu không sẽ không làm được gì khi hợp đồng đã ký kết. Mặt khác, quy mô dự án phải đủ lớn để chịu được các chi phí trả chuyên gia phân tích rủi ro. Boehm cải biên mô hình này năm 1998 với tên là "*WINWIN Spiral Model*" dùng cho việc phát triển phần mềm hướng đối tượng.

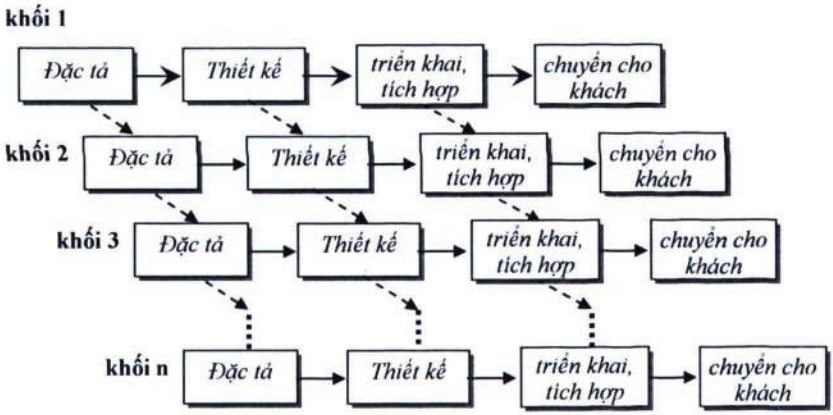
2.3.3. Mô hình RUP

Mô hình RUP *(Rational Unified Process)* là mô hình phát triển phần mềm dành riêng cho hướng đối tượng. Nó có ba đặc trưng: *Lấy kiến trúc làm trung tâm, điều khiển bởi các ca sử dụng, lặp và tăng dần.* Nó có nhiều điểm tương đồng với mô hình xoắn ốc được Boehm (1998) [16] làm thích ứng cho việc phát triển phần mềm hướng đối tượng. Tuy nhiên, ở mỗi bước lặp của RUP, nội dung các hoạt động có những đặc trưng riêng gắn với ngôn ngữ mô hình hóa trực quan - UML.

Chính bản thân UML không định nghĩa tiến trình phát triển phần mềm, nhưng UML và phần mềm công cụ *Rational Rose* (gọi tắt là *Rose*) được sử dụng có hiệu quả trong một số công đoạn của tiến trình phát triển phần mềm [5].

2.3.4. Mô hình phát triển tăng dần

Mô hình phát triển tăng dần (Incremental model): phát triển một hệ thống càng nhanh càng tốt, rồi cải biên hệ thống đó cho tới khi nó thực hiện được mọi yêu cầu đặt ra. Nó là một biến thể của mô hình tiến hóa. Nó có ý tưởng cơ bản gần giống với mô hình làm bản mẫu và mô hình xoắn ốc, nhưng được thực hiện trên từng khối tương đối độc lập (hình 2.5).



Hình 2.5. Mô hình phát triển tăng dần [16]

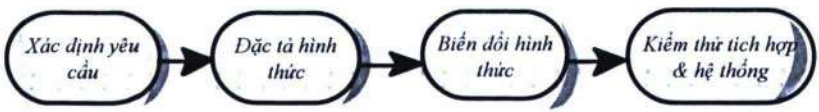
2.3.5. Mô hình phát triển ứng dụng nhanh RAD

RAD *(Rapid Application Development)* là phương pháp luận *gộp các hoạt động phân tích, thiết kế, xây dựng vào một loạt vòng lặp phát triển ngắn*: bước lặp ngắn bảo đảm tính hiệu quả và tự sửa chữa. Kỹ thuật RAD hướng đến nhu cầu đưa người dùng tham gia vào việc phân tích và thiết kế hệ thống bằng cách sử dụng CASE. Hệ thống phát triển bằng RAD đáp ứng nhu cầu hiệu quả và chi phí bảo trì thấp. Chất lượng hệ thống được xác định ở đây là mức độ hệ thống đáp ứng các yêu cầu nghiệp vụ về thời gian.

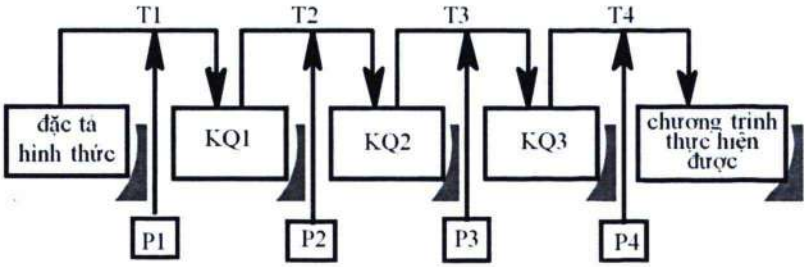
RAD thích hợp cho đội phát triển nhỏ, người dùng và nguồn lực của HTTT không lớn, kết hợp với vòng lặp ngắn, tối ưu tốc độ, thống nhất cái nhìn và mục tiêu giao tiếp không hình thức, hiệu quả và quản lý dự án đơn giản, mỗi bước xuất ra một phiên bản chức năng của hệ thống cuối cùng. Nó được kỹ nghệ riêng, một phần làm việc đầy đủ, không giống như làm mẫu.

2.3.6. Mô hình phát triển hệ thống hình thức hóa

Quá trình phát triển hệ thống hình thức hóa được mô tả trên hình 2.6. Tư tưởng chính của phương pháp này là biểu diễn các đặc tả yêu cầu ở dạng biểu diễn các ký pháp toán học. Sau đó áp dụng các biến đổi khác nhau để chuyển đặc tả hình thức thành chương trình. Trong quá trình chuyển đổi, biểu diễn hình thức của đặc tả ngày càng được chi tiết dần nhưng luôn bảo đảm tính đúng đắn toán học của biểu diễn hệ thống mà không cần đòi hỏi một sự kiểm tra tốn kém. Nhờ vậy mà nó bảo đảm rằng chương trình nhận được là một triển khai đúng của đặc tả.



Hình 2.6. Các bước của tiến trình phát triển hình thức [16]



Các chứng minh sự đúng đắn của phép biến đổi

Hình 2.7. Các bước của tiến trình phát triển biến đổi hình thức [16]

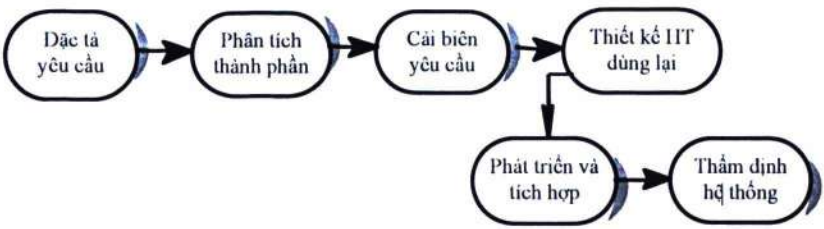
Ưu điểm của cách tiếp cận này là có thể áp dụng để chứng minh tính đúng đắn của đặc tả, chứng minh được chương trình là đáp ứng được yêu cầu của đặc tả đã cho. Chi phí cho đặc tả cao, nhưng các chi phí sau đó là nhỏ hơn nhiều so với các phương pháp khác. Quá trình chuyển đổi tạo thành một dãy các bước nhỏ và có thể dễ theo dõi được. Tuy nhiên, việc đặc tả đòi hỏi một trình độ trừu tượng cao, việc chọn các chuyển đổi nào để áp dụng lại yêu cầu kỹ năng rất chuyên nghiệp, việc chứng minh sự đúng đắn của chuyển đổi là rất khó khăn. Vì vậy, với đa số các kỹ sư phần mềm, phương pháp này là khó áp dụng.

2.3.7. Mô hình phát triển phần mềm theo hướng sử dụng lại

Mô hình phát triển phần mềm theo hướng sử dụng lại dựa trên cơ sở có rất nhiều các thành phần mềm có thể sử dụng và tích hợp chúng trong một khung làm việc để nhận được một hệ thống làm việc.

Sơ đồ tổng quát của quá trình được mô tả như ở hình 2.8. Trước hết, giống như các tiến trình khác cần đặc tả yêu cầu, sau đó phân tích các yêu cầu thành những thành phần. Tiến hành cải biên các thành phần này để chúng tương ứng với các thành phần sẵn có. Sau đó thiết kế hệ thống dựa trên các thành phần đã có. Trong đó đặc biệt có thể sử dụng các mẫu thiết kế (patterns). Công việc tiếp theo là phát triển và tích hợp hệ thống. Trong trường hợp chung nhất, khi còn có những thành phần chưa có sẵn, lúc đó cần phải phát triển bổ sung. Trong trường hợp đặc biệt, các tài sản có sẵn là những phần mềm, khi đó

ta có phương pháp phát triển dựa trên thành phần (CBSE - Component-Based Software Engineering).



Hình 2.8. Tiến trình phát triển hướng sử dụng lại [16]

Phương pháp này có ưu điểm vượt trội là rút ngắn được thời gian phát triển, giảm chi phí và đặc biệt ít gặp rủi ro. Hơn thế nữa nó còn cho hệ thống có độ tin cậy cao. Ngày nay, công nghệ hướng đối tượng phát triển phần mềm tiến triển rất nhanh, phát triển phần mềm theo hướng sử dụng lại được sự trợ giúp rất lớn từ công nghệ này. Đặc biệt là dùng lại *các khung làm việc* (*Framework*) và *mẫu* thiết kế.

2.3.8. Mô hình phát triển phần mềm theo kỹ thuật thế hệ thứ tư

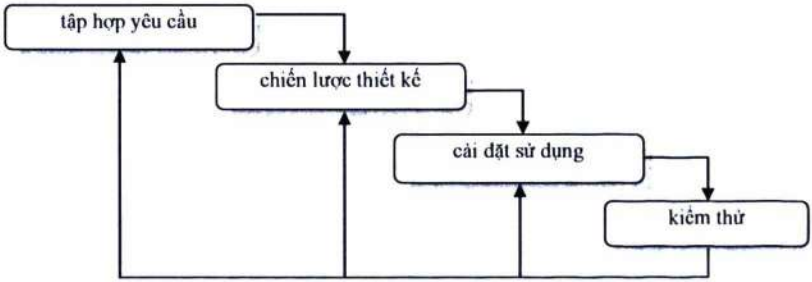
Thuật ngữ "*kỹ thuật thế hệ thứ 4*" (4GT) bao gồm một phạm vi rộng các công cụ phần mềm có điểm chung là: Mỗi công cụ đều cho phép xác định một số đặc trưng của phần mềm ở mức cao. Tiếp đó, công cụ tự động sinh ra mã chương trình gốc theo nhu cầu của người phát triển. Tiến trình 4GT tập trung vào khả năng xác định phần mềm ở mức độ gần với ngôn ngữ tự nhiên hay dùng một số ký pháp tương ứng với chức năng. Tiến trình 4GT cho kỹ nghệ phần mềm được thể hiện trên hình 2.9.

Môi trường phát triển phần mềm hỗ trợ cho tiến trình 4GT bao gồm một số hoặc tất cả các công cụ sau:

- Ngôn ngữ phi thủ tục để truy vấn CSDL,
- Bộ sinh báo cáo,
- Bộ thao tác dữ liệu,
- Bộ tương tác và xác định màn hình,

- Bộ sinh chương trình/mã,

- Khả năng làm trang tính.



Hình 2.9. Mô hình tiến trình sử dụng ngôn ngữ thế hệ thứ 4

2.4. Các hoạt động của tiến trình phát triển phần mềm

Các hoạt động cơ bản của tiến trình phát triển phần mềm bao gồm:

1- Đặc tả yêu cầu phần mềm và yêu cầu hệ thống (xác định và đặc tả các chức năng của phần mềm và các ràng buộc mà nó cần tuân thủ khi phát triển và hoạt động);

2- Đặc tả thiết kế (đặc tả các chức năng của phần mềm và các ràng buộc ở mức cao, gần với mức cài đặt);

3- Lập trình và thử nghiệm (tạo ra phần mềm theo đúng yêu cầu và thiết kế đã được đặc tả);

4- Kiểm thử phần mềm (thẩm định phần mềm xem có đáp ứng yêu cầu người dùng không);

5- Đánh giá chất lượng phần mềm (bảo đảm chất lượng phần mềm);

6- Nâng cấp phần mềm (phần mềm được tiến hóa để đáp ứng yêu cầu thay đổi của người dùng và môi trường).

Trong các tiến trình khác nhau, các hoạt động này được tổ chức theo những cách khác nhau và được mô tả theo những mức chi tiết khác nhau. Tuy nhiên, mỗi kiểu tiến trình có thể là thích hợp với các loại ứng dụng này hơn là với loại khác.

Lần lượt trong các chương tiếp theo, chúng ta sẽ nghiên cứu kỹ lưỡng các nội dung của các hoạt động này. Ở đây, cần liệt kê các tài liệu đưa ra cho các hoạt động đó.

Bảng 2.1 trình bày các hoạt động cụ thể và các tài liệu tương ứng cần có trong quá trình phát triển được kiểm soát, lưu trữ trong cấu hình phần mềm.

Bảng 2.1. Các hoạt động phát triển phần mềm và các tài liệu tương ứng

Các hoạt động		***Tài liệu đưa ra***
1- Đặc tả yêu cầu phần mềm và yêu cầu hệ thống	- Phân tích yêu cầu	- Báo cáo nghiên cứu khả thi; - Yêu cầu khái quát.
	- Xác định yêu cầu	Tài liệu yêu cầu.
	- Đặc tả hệ thống	- Đặc tả chức năng; - Kế hoạch kiểm thử chấp nhận; - Giới thiệu tóm tắt cách sử dụng.
	- Đặc tả phần mềm	- Cấu trúc các hệ con và giao diện; - Mô tả các hệ con phần mềm.
2- Đặc tả thiết kế	- Thiết kế kiến trúc	- Đặc tả kiến trúc; - Kế hoạch kiểm thử hệ thống.
	- Thiết kế dữ liệu	- Cấu trúc dữ liệu lôgic/vật lý; - Từ điển dữ liệu.
	- Thiết kế giao diện	- Đặc tả giao diện; - Kế hoạch kiểm thử tích hợp.
	- Thiết kế chi tiết	- Đặc tả thiết kế; - Kế hoạch kiểm thử đơn vị.
3- Lập trình và thử nghiệm	- Mã hóa và thử nghiệm	- Chương trình nguồn; - Biên bản thử nghiệm.

4- Kiểm thử phần mềm	- Kiểm thử đơn vị	Báo cáo kiểm thử đơn vị.
	- Kiểm thử mô-đun	Báo cáo kiểm thử mô đun.
	- Kiểm thử tích hợp	- Báo cáo kiểm thử tích hợp; - Hướng dẫn sử dụng cuối cùng.
	- Kiểm thử hệ thống	Báo cáo kiểm thử hệ thống.
	- Kiểm thử chấp nhận	Hệ thống chương trình và các tài liệu.
5- Bảo đảm chất lượng phần mềm	- Thẩm định và xác minh	- Báo cáo các khuyết tật trong hệ thống và tính khả dụng của hệ thống.
	- Kiểm thử phần mềm	- Các chiến lược kiểm thử đề xuất; - Các kế hoạch và các đặc tả thiết kế ca kiểm thử; - Đánh giá các lỗi phần mềm phát hiện được sau kiểm thử.
	- Quản lý cấu hình phần mềm	- Sổ quản lý các "chế tác" trong tiến trình kỹ nghệ phần mềm.
	- Các hoạt động quản lý chất lượng khác	Các báo cáo kết quả: áp dụng công nghệ, kỹ nghệ hiệu quả; rà soát kỹ chính thức và rà soát phần mềm; tuân theo các chuẩn phát triển; khống chế thay đổi; thực hiện đo lường; báo cáo và quản lý các báo cáo; gỡ lỗi; phân tích an toàn phần mềm.
6- Nâng cấp phần mềm	- Thẩm định yêu cầu nâng cấp; - Thực hiện nâng cấp.	- Yêu cầu nâng cấp; - Báo cáo kết quả nâng cấp.

TÓM TẮT CHƯƠNG

Tiến trình phần mềm là một loạt các bước hoạt động dự kiến trước, mà theo đó ta có thể tạo ra một phần mềm chất lượng, hiệu quả và đúng thời hạn. Đặc trưng của tiến trình là các hoạt động cần thực hiện bao gồm các hoạt động mang tính kỹ thuật để tạo ra sản phẩm và trình tự thực hiện các hoạt động.

Mô hình tiến trình là một sự đơn giản hóa một lớp các tiến trình thực. Một vài mô hình tiến trình phần mềm được nhiều người biết đến như: mô hình thác nước, mô hình xoắn ốc, mô hình làm bản mẫu,...

Ý tưởng chung của phương pháp tiến hóa là phát triển một phiên bản sơ khai ban đầu và giới thiệu để người dùng bình luận rồi làm mịn nó thông qua các bước cho đến khi nhận một hệ thống thích hợp.

Cách tiếp cận làm bản mẫu cho kỹ nghệ phần mềm là cách tiếp cận thực tế nhất. Nó rất thích hợp với các hệ thống vừa và nhỏ, đặc biệt khi yêu cầu chưa rõ ràng, chưa xác định cụ thể cái vào cái ra và chưa chắc chắn nội dung xử lý hay hiệu quả của một thuật toán.

Mô hình xoắn ốc dựa trên ý tưởng là tối thiểu hóa rủi ro bằng việc phân tích yếu tố rủi ro ở mỗi bước lặp và sử dụng phương pháp làm bản mẫu. Mô hình này thích hợp để phát triển các hệ thống quy mô lớn.

Mô hình RUP là mô hình phát triển phần mềm dành riêng cho hướng đối tượng. Nó có ba đặc trưng: Lấy kiến trúc làm trung tâm; Điều khiển bởi các ca sử dụng; Lặp và tăng dần.

Mô hình phát triển tăng dần là phát triển một hệ thống càng nhanh càng tốt, rồi cải biên hệ thống đó cho tới khi nó thực hiện được mọi yêu cầu đặt ra. Nó là một biến thể của mô hình tiến hóa.

RAD là phương pháp luận gộp các hoạt động phân tích, thiết kế, xây dựng vào một loạt vòng lặp phát triển ngắn: bước lặp ngắn đảm bảo tính hiệu quả và tự sửa. Kỹ thuật RAD hướng đến nhu cầu đưa người dùng tham gia vào việc phân tích và thiết kế hệ thống bằng cách sử dụng CASE.

Mô hình phát triển phần mềm theo hướng sử dụng lại dựa trên cơ sở có rất nhiều các thành phần mềm có thể sử dụng và tích hợp chúng trong một khung làm việc để nhận được một hệ thống làm việc.

4GT có điểm chung: mỗi công cụ đều cho phép xác định một số đặc trưng của phần mềm ở mức cao; công cụ tự động sinh ra mã chương trình gốc theo nhu cầu của người phát triển. 4GT tập trung vào khả năng xác định phần mềm ở mức độ gần với ngôn ngữ tự nhiên hay dùng một số ký pháp tương ứng với chức năng.

Có 6 hoạt động cơ bản của tiến trình phát triển phần mềm: Đặc tả yêu cầu phần mềm và yêu cầu hệ thống; Đặc tả thiết kế; Lập trình và thử nghiệm; Kiểm thử phần mềm; Bảo đảm chất lượng phần mềm; Nâng cấp phần mềm.

CÂU HỎI CHƯƠNG 2

2.1. Tại sao chúng ta phải quan tâm đến tiến trình phát triển phần mềm?

2.2. Tiến trình, quá trình và quy trình có gì khác nhau?

2.3. Tiến trình và mô hình tiến trình có những nét gì khác biệt?

2.4. Các pha trong mô hình thác nước bao gồm hoạt động nào?

2.5. Tại sao mô hình thác nước tuy là mô hình cổ điển (ra đời sớm nhất) nhưng đến nay nó vẫn còn giá trị?

2.6. Theo Anh/Chị thì mô hình tiến hóa và mô hình thác nước khác nhau cơ bản ở điểm nào?

2.7. Cách tiếp cận làm bản mẫu cho kỹ nghệ phần mềm là cách tiếp cận thực tế nhất. Nó rất thích hợp với các hệ thống có quy mô như thế nào?

2.8. Mô hình xoắn ốc thích hợp để phát triển các hệ thống có quy mô như thế nào?

2.9. Phương pháp dựa theo hướng sử dụng lại có ưu điểm gì?

2.10. Trong 6 hoạt động của tiến trình (1-Đặc tả yêu cầu phần mềm và yêu cầu hệ thống; 2-Đặc tả thiết kế; 3-Lập trình và thử nghiệm; 4-Kiểm thử phần mềm; 5-Bảo đảm chất lượng phần mềm; 6-Nâng cấp phần mềm) thì hoạt động nào, theo Anh/Chị, là quan trọng nhất? tại sao?

Chương 3
ĐẶC TẢ YÊU CẦU PHẦN MỀM VÀ YÊU CẦU HỆ THỐNG

Yêu cầu phần mềm và yêu cầu hệ thống phải được viết sao cho NSD có thể hiểu được mà không cần một kiến thức chuyên môn đặc biệt nào.

Chương này về các yêu cầu phần mềm và hệ thống và các kỹ thuật thể hiện yêu cầu.

Học viên cần nắm vững các yêu cầu hệ thống, yêu cầu phần mềm, tiến trình xác định yêu cầu và đặc tả yêu cầu phần mềm. Học viên cần rèn luyện các kỹ năng đặc tả yêu cầu và viết tài liệu yêu cầu.

3.1. Các yêu cầu hệ thống

3.1.1. Khái niệm về yêu cầu hệ thống

Yêu cầu cho một hệ thống mô tả những việc mà hệ thống làm và những ràng buộc mà nó phải tuân thủ khi hoạt động [16].

Ví dụ 3.1

Đặc tả yêu cầu hệ thống:

- Người dùng phải được cung cấp phương tiện để dễ dàng xác định được kiểu file ngoài.

- Với mỗi kiểu file ngoài cần có một công cụ thích hợp cho nó.

- Mỗi kiểu file ngoài có thể được thể hiện bằng một biểu tượng (icon) trên màn hình của người dùng.

- Phương tiện phải được cung cấp cho mỗi biểu tượng đại diện của một kiểu file ngoài khi người dùng xác định.

- Khi người dùng chọn một biểu tượng cho kiểu file ngoài thì công cụ thích hợp được áp dụng để truy nhập và trình diễn kiểu file đó.

3.1.2. Phân loại yêu cầu hệ thống

Yêu cầu hệ thống được phân thành 4 loại cơ bản:

1- Các yêu cầu chức năng

Các yêu cầu chức năng (functional requirements) là các yêu cầu về các dịch vụ mà phần mềm /hệ thống phải cung cấp. Yêu cầu chức năng hiểu theo nghĩa rộng cũng có thể là yêu cầu không nên thực hiện chức năng nào đó.

Các yêu cầu chức năng cho một hệ thống *được thể hiện bằng các đặc tả về các hoạt động của hệ thống hay các dịch vụ mà hệ thống sẽ cung cấp*. Chúng phụ thuộc vào loại phần mềm được xây dựng, vào sự mong đợi của người dùng, vào quy mô hệ thống được phát triển và môi trường mà ở đó nó được cài đặt (môi trường đích).

Về nguyên tắc, đặc tả yêu cầu chức năng cần phải *đầy đủ* và *nhất quán*. Đầy đủ, có nghĩa là tất cả chức năng mà người dùng cần ở hệ thống phải được xác định. Nhất quán nghĩa là các yêu cầu nêu ra không được mâu thuẫn với nhau. Tuy nhiên, đối với một hệ thống lớn và phức tạp, điều này rất khó có thể đạt ngay được.

2- Các yêu cầu phi chức năng

Các yêu cầu phi chức năng (non-functional requirements) là các ràng buộc mà phần mềm/hệ thống phải tuân theo. Đó là những ràng buộc bắt các dịch vụ hoặc chức năng mà hệ thống cung cấp phải tuân thủ.

Các yêu cầu phi chức năng không liên quan trực tiếp tới các hoạt động chức năng cụ thể của hệ thống. Chúng có thể liên quan đến các thuộc tính quan trọng của hệ thống, đến hiệu năng của hệ thống như *độ tin cậy, thời gian đáp ứng của dịch vụ, việc sử dụng bộ nhớ*, hay đến những *ràng buộc lên hệ thống* như khả năng vào ra của thiết bị, sự biểu diễn dữ liệu trên các giao diện của hệ thống,... Chúng bao gồm các ràng buộc như về thời gian, về ngân sách, các ràng buộc trong tiến trình phát triển, hay các ràng buộc về chuẩn sử dụng,..

Rất nhiều yêu cầu phi chức năng liên quan đến tổng thể hệ thống hơn là đến từng bộ phận riêng rẽ. Điều đó có nghĩa là, chúng thường quan trọng hơn

rất nhiều những chức năng bộ phận, trong khi lỗi của một chức năng bộ phận có thể sửa đổi và nâng cấp, thì lỗi của một yêu cầu phi chức năng có thể làm cho hệ thống ngừng hoạt động, không sử dụng được.

Sommerville [25] đã mô tả các kiểu khác nhau của yêu cầu phi chức năng:

- *Yêu cầu về sản phẩm:* yêu cầu này xuất hiện từ những đòi hỏi của sản phẩm sẽ được phân phối sử dụng trong những trường hợp cụ thể. Ví dụ yêu cầu về thời gian hệ thống phải trả lại kết quả, về tính tin cậy (tỷ lệ lỗi chấp nhận được),... Các yêu cầu này thường trực tiếp từ nhu cầu của NSD.

- *Yêu cầu mang tính tổ chức:* Những yêu cầu này xuất phát từ các quy định và thủ tục của các tổ chức. Ví dụ, một số chuẩn bắt buộc phải áp dụng như *yêu cầu ngôn ngữ lập trình* hoặc *phương pháp thiết kế* được sử dụng,... Các yêu cầu này có thể từ cả NSD và từ các tổ chức phát triển.

- *Yêu cầu mở rộng:* Khái niệm này để chỉ tất cả những yêu cầu xuất phát từ các nhân tố bên ngoài tác động lên hệ thống trong tiến trình phát triển và vận hành. Nó bao gồm yêu cầu về *tính tương hợp:* xác định cách thức hệ thống tương tác với các hệ thống khác trong môi trường mà nó hoạt động. Yêu cầu về *tính pháp lý*, yêu cầu về *tính văn hóa* đặt ra nhằm bảo đảm sản phẩm đưa ra phải phù hợp với phong tục, đạo đức, thói quen của người dùng,...

3- Các yêu cầu miền lĩnh vực

Trong thực tế, người ta còn quan tâm đến *các yêu cầu miền lĩnh vực (domain requirements).* Đó là các yêu cầu xuất phát từ miền ứng dụng. Chúng phản ánh các đặc trưng của lĩnh vực ứng dụng này thông qua các yêu cầu nghiệp vụ. Chúng có thể là chức năng hay phi chức năng.

4- Các yêu cầu của người dùng

Yêu cầu của người dùng (User requirements) là những đòi hỏi của NSD về các dịch vụ mà hệ thống cung cấp và các ràng buộc khi nó hoạt động. Chúng được diễn tả bằng ngôn ngữ tự nhiên và thường có kèm theo biểu đồ. Đó là *những mô tả theo cách của người dùng.*

Ví dụ về yêu cầu người dùng: Phần mềm phải cung cấp phương tiện biểu diễn và truy cập tới những file ngoài được tạo ra bởi một công cụ khác.

3.1.3. Các cách tiếp cận và các kỹ thuật phân tích yêu cầu hệ thống

3.1.3.1. Khái quát về các cách tiếp cận và các kỹ thuật phân tích

Có nhiều phương pháp khác nhau để tiếp cận việc xác định yêu cầu:

1- Phương pháp mô hình. Đây là phương pháp phân tích yêu cầu được sử dụng rộng rãi nhất. Một số mô hình được xây dựng để dành riêng cho việc suy luận và phân tích yêu cầu. Một số khác lại rất gần với việc định nghĩa cho một phương pháp. Các phân tích dựa trên mô hình điển hình đi theo hai hướng tiếp cận:

+ *Tiếp cận định hướng chức năng* là các phương pháp phân tích hướng cấu trúc dựa trên luồng dữ liệu.

+ *Tiếp cận định hướng đối tượng.*

2- Phương pháp hướng cách nhìn *(viewpoint)*. Sommerville (2001), đã ghi nhận hướng *cách nhìn* theo một trong các góc độ sau:

+ *Từ nguồn hay đích tới của dữ liệu*: Cách nhìn ở đây tập trung vào điểm đại diện cho việc tạo ra hay sử dụng dữ liệu. Việc phân tích bao gồm việc xác định dữ liệu gì được tạo ra, được sử dụng và những xử lý nào được tiến hành. Nó điển hình cho phương pháp phân tích vào / ra (*Input - Output analysis*).

+ *Từ khung làm việc trình diễn:* Cách nhìn này xem xét từ loại hình công cụ mô tả mô hình hệ thống. Ví dụ như mô hình *thực thể - mối quan hệ, mô hình máy trạng thái hữu hạn.* Mỗi cách tiếp cận phát hiện ra những đối tượng khác nhau của hệ thống được phân tích và những mối quan hệ của chúng.

+ *Từ sự tiếp nhận dịch vụ:* Cách nhìn này từ ngoài vào hệ thống và sự tiếp nhận dịch vụ từ hệ thống. Các quan điểm này cung cấp dữ liệu cho các dịch vụ và các tín hiệu kiểm tra. Việc phân tích bao gồm kiểm tra các dịch vụ nhận được theo quan điểm khác nhau, thu thập chúng và giải quyết các xung đột.

Có nhiều kỹ thuật phân tích yêu cầu. Một số kỹ thuật điển hình:

1- Kỹ thuật phân tích yêu cầu dựa trên phương pháp mô hình hoá

Kỹ thuật phân tích yêu cầu dựa trên phương pháp mô hình hoá đang là phương pháp phổ dụng hiện nay. Chúng ta sẽ tìm hiểu phương pháp này kỹ hơn thông qua những ví dụ được trình bày dưới đây.

2- Kỹ thuật VORD *(Viewpoint-Oriented Requirements Definition)* được Kotonua and Sommeville đề xuất năm 1996. Các giai đoạn cơ bản của VORD bao gồm:

+ *Xác định khung nhìn* bao gồm việc tìm kiếm các khung nhìn thu nhận các dịch vụ hệ thống và việc xác định các dịch vụ cung cấp cho từng khung nhìn.

+ *Cấu trúc khung nhìn* bao gồm nhóm các khung nhìn liên quan vào trong một hệ thống phân cấp. Các dịch vụ chung được cung cấp ở mức cao trong hệ thống này và được các khung nhìn ở mức thấp hơn kế thừa.

+ *Làm tài liệu khung nhìn* bao gồm việc làm mịn các mô tả khung nhìn và dịch vụ được xác định.

+ *Ánh xạ hệ thống - khung nhìn:* xác định các đối tượng trong một thiết kế hướng đối tượng khi sử dụng các thông tin dịch vụ được bao gói trong một khung nhìn.

3- Kỹ thuật phân tích hình thức hoá

Kỹ thuật phân tích hình thức hóa dựa trên việc sử dụng các khái niệm, ký pháp và mô hình toán học để phân tích và biểu diễn hệ thống. Việc phân tích không tách thành mô hình riêng, mà kết quả của hoạt động phân tích và mô hình hóa cho ngay đặc tả yêu cầu của hệ thống.

3.1.3.2. Kỹ thuật phân tích yêu cầu dựa trên phương pháp mô hình hoá

Để phát hiện được các yêu cầu, cả hai cách tiếp cận hướng chức năng và hướng đối tượng đều tập trung hướng vào mô tả mặt nghiệp vụ của hệ thống thực và kết quả thu được là một *mô hình nghiệp vụ.*

1- Mô hình nghiệp vụ

Mô hình nghiệp vụ được xây dựng trên cơ sở của phương pháp phân tích hệ thống bao gồm:

+ *Một mô hình ngữ cảnh* mô tả hệ thống được xét trong môi trường của nó. Ở đây đặc biệt chú ý đến các yếu tố môi trường có tương tác với hệ thống

gọi là các *tác nhân*. Thông qua mô hình này cũng cho phép xác định rõ phạm vi của hệ thống cần nghiên cứu. Tùy theo cách tiếp cận mà khái niệm về *tác nhân* và *mối quan hệ tương tác* của hệ thống với tác nhân được mô tả bằng cách khác nhau (cả về khái niệm và ký pháp).

+ *Các mô hình cấu trúc chức năng* mô tả cấu trúc các chức năng bên trong hệ thống. Các mô hình này thường có nhiều mức, mỗi mức mô tả chức năng ở mức gộp hay chi tiết khác nhau. Ở mức thấp nhất, nội dung các chức năng đủ dễ hiểu và có thể mô tả chi tiết nó với các thông tin cần thiết. Tùy theo cách tiếp cận mà khái niệm chức năng và liên kết cấu trúc giữa chúng cũng như ký pháp sử dụng là khác nhau. Ở đây các phương pháp phân hoạch/phân rã các chức năng khác nhau được sử dụng.

+ *Mô tả chi tiết các chức năng*: Một cách tự nhiện, mô tả chi tiết chỉ cần thiết và có thể thực hiện với các chức năng mức thấp nhất - chức năng cơ sở. Tuy nhiên, cũng có thể mô tả chức năng ở mức cao (theo hướng đối tượng) để dễ hiểu và lần vết trong quá trình tiếp cận dần. Tùy thuộc vào cách tiếp cận, nội dung và cách mô tả chức năng cũng khác nhau.

+ *Mô tả các đối tượng dữ liệu* cần cho việc lưu giữ sau này. Do bản chất của mỗi cách tiếp cận mà sự mô tả ở đây là khác nhau về cơ bản. Theo hướng cấu trúc, mô tả dữ liệu bao gồm tất cả hồ sơ được sử dụng trong hoạt động nghiệp vụ và các bản mẫu thực của chúng. Trong khi đó, theo hướng đối tượng, mô tả lại bao gồm các đối tượng và khái niệm của thế giới thực.

+ *Mô tả mối liên kết giữa dữ liệu và chức năng:* Mô tả này chỉ cần thiết đối với cách tiếp cận hướng cấu trúc.

+ *Từ điển giải thích* liên quan đến các khái niệm và dữ liệu sử dụng. Từ điển này là cần thiết để có sự hiểu biết chung thống nhất về hệ thống thực được mô tả cũng như các đặc trưng của các dữ liệu cần lưu giữ.

2- Các thể hiện của mô hình phân tích

Mô hình nghiệp vụ của hoạt động phân tích yêu cầu theo hai cách tiếp cận định hướng cấu trúc và định hướng đối tượng được thể hiện trong bảng 3.1.

Bảng 3.1. Thể hiện của mô hình nghiệp vụ theo cách tiếp cận

Các thành phần của mô hình	**Thể hiện của mô hình theo**	
	Hướng cấu trúc	***Hướng đối tượng***
1- *Mô hình ngữ cảnh*	Gồm ba thành phần: - Các *tiến trình* (*hình chữ nhật góc tròn/elip*) duy nhất mô tả hệ thống, - Các *tác nhân (hình chữ nhật)* mô tả người, tổ chức, hệ thống khác không thuộc hệ thống - Các *luồng dữ liệu* (mũi tên) liên kết giữa tác nhân và hệ thống, kèm theo các thông tin chuyển giao	Gồm ba thành phần: - Các *ca sử dụng* (*hình elip*) mức cao mô tả hệ thống - Các *tác nhân* (*hình người*) là người, các hệ khác có tương tác với hệ thống - *Các liên kết tương tác* (*ký pháp UML*) giữa tác nhân với các ca sử dụng, và giữa ca sử dụng với nhau
2- *Mô hình cấu trúc các chức năng*	Gồm một hay một số *biểu đồ phân cấp hình cây*, mỗi nút là một chức năng; Mỗi chức năng ở mức dưới nhận được bằng cách phân rã chức năng trên của nó; Chức năng mức thấp nhất là chức năng cơ sở .	Là các *biểu đồ ca sử dụng* tương ứng với các mức khác nhau; Biểu đồ của mức dưới nhận được từ sự phân rã các ca sử dụng mức trên; Ca sử dụng mức thấp có thể mô tả sự tương tác.
3- *Mô tả chi tiết chức năng*	Chỉ mô tả chức năng cơ sở; Mô tả gồm tên chức năng, các dữ liệu vào, dữ liệu ra, nguồn, đích, các quy tắc nghiệp vụ để thực hiện chức năng.	Mô tả mỗi ca sử dụng bằng các luồng tương tác chính/ ngoại lệ giữa tác nhân và hệ thống; Các ràng buộc lên luồng tương tác.

4- *Mô tả các đối tượng dữ liệu*	*Danh sách các hồ sơ* dữ liệu sử dụng; Các mẫu hồ sơ, từ điển dữ liệu.	*Mô hình miền lĩnh vực* với các đối tượng là các thực thể hay khái niệm được liên kết với nhau.
5- *Mô tả liên kết giữa chức năng và dữ liệu*	Ma trận thực thể-chức năng: mỗi cột ứng với một hồ sơ, mỗi dòng-một chức năng; Mỗi ô ghi tương tác giữa chức năng và hồ sơ.	Không cần
6- *Từ điển giải thích*	Từ điển giải thích	Từ điển giải thích khái niệm

3.2. Các yêu cầu phần mềm

Yêu cầu phần mềm là các đòi hỏi không thể thiếu về các chức năng */dịch vụ* mà phần mềm dự kiến phải cung cấp và các *ràng buộc* mà phần mềm và cả hệ thống phải tuân thủ khi vận hành. Yêu cầu phần mềm phải được viết sao cho NSD có thể hiểu được mà không cần một kiến thức chuyên môn đặc biệt nào. Yêu cầu phần mềm được biểu diễn bằng các mô tả trừu tượng.

Có hai đòi hỏi cơ bản để đối với các yêu cầu phần mềm:

1- *Các yêu cầu phần mềm phải bộc lộ theo phương thức "trên- xuống". Các chức năng, giao diện và thông tin chủ yếu phải được hiểu hoàn toàn rõ trước khi xác định chi tiết các tầng kế tiếp.*

2- Người phân tích cũng phải hiểu từng khuôn cảnh phần mềm và đánh giá được các bước kỹ nghệ phần mềm tổng quát, áp dụng được bất kể khuôn cảnh nào cần dùng. Nhiều yêu cầu phần mềm không tường minh (như thiết kế cho bảo trì) được tổ hợp vào Bản đặc tả yêu cầu *chỉ nếu người phân tích hiểu được kỹ nghệ phần mềm.*

Bản đặc tả các yêu cầu phần mềm được tạo ra tại đỉnh cao của nhiệm vụ phân tích.

Chức năng và hiệu năng được cấp phát cho phần mềm xem như một phần của kỹ nghệ hệ thống thường được làm mịn bằng cách thiết lập phần mô tả

thông tin đầy đủ như *mô tả chức năng chi tiết, chỉ báo về các yêu cầu hiệu năng và những ràng buộc thiết kế, các tiêu chuẩn hợp lệ thích hợp và những dữ liệu khác thường cần tới*. Cơ quan tiêu chuẩn quốc gia, IEEE và Bộ quốc phòng Mỹ đề nghị các định dạng cho các đặc tả yêu cầu phần mềm.

Bảng 3.2. Dàn bài đặc tả yêu cầu phần mềm [23]

Dàn bài đặc tả yêu cầu phần mềm	
I. Giới thiệu	
A. Đại cương về hệ thống	(mục tiêu, mục đích)
B. Mô tả chung	(cấu trúc)
C. Các ràng buộc dự án phần mềm	(phạm vi)
II. Mô tả thông tin	(chi tiết)
A. Biểu diễn luồng thông tin	(dựa vào cấu trúc thông tin)
1. Luồng dữ liệu	
2. Luồng điều khiển	(đặc biệt chú ý đến khi xét các hệ thời gian thực)
B. Biểu diễn nội dung thông tin	
C. Mô tả giao diện hệ thống	
III. Mô tả chức năng	
A. Phân hoạch chức năng	(cho từng chức năng)
B. Mô tả chức năng	
1. Tường thuật về cách xử lý	(lời giải thích)
2. Hạn chế/giới hạn	
3. Yêu cầu hiệu năng	(về các mặt, kinh tế, xã hội, kỹ thuật, chi phí, thời gian)
4. Ràng buộc thiết kế	(theo từng mô-đun)
5. Biểu đồ trợ giúp	(có luận giải)
	(cấu trúc tổng thể, tương quan với các phần từ hệ thống khác)

C. Mô tả điều khiển	
1. Đặc tả điều khiển	
2. Ràng buộc thiết kế	(có luận giải)
IV. Mô tả hành vi	(xem xét vận hành của phần mềm)
A. Trạng thái hệ thống	(mức toàn cảnh, mức đỉnh)
B. Sự kiện và hành động	(mức cụ thể, bộ phận)
V. Tiêu chuẩn hợp lệ	(quan trọng nhất nhưng hay bị bỏ quên)
A. Giới hạn hiệu năng	
B. Lớp các kiểm thử	(tiêu chuẩn hướng tới (lí tưởng))
C. Đáp ứng phần mềm dự kiến	(tài liệu công nghệ phần mềm khác, tham khảo kỹ thuật ,...)
D. Các xem xét đặc biệt	(Danh sách các nhà cung cấp, các chuẩn,...)
VI. Sách tham khảo	
VII. Phụ lục	

Trong nhiều trường hợp bản *đặc tả các yêu cầu Phần mềm* còn có thể còn có kèm theo một bản mẫu thực hiện được (mà trong một số trường hợp có thể thay thế được cho bản đặc tả), một bản mẫu trên giấy, hay *tài liệu sơ bộ của người dùng*. Bản *tài liệu sơ bộ của người* dùng trình bày phần mềm như hộp đen. Tức là, chủ yếu nhấn mạnh vào cái vào của người dùng và cái ra kết quả. Tài liệu này có thể dùng như một công cụ có giá trị để làm lộ ra vấn đề ở giao diện người -máy.

Việc xét duyệt bản *đặc tả các yêu cầu phần mềm* (và hoặc bản mẫu) do cả người phát triển phần mềm và khách hàng cùng tiến hành. Bởi vì đặc tả tạo nên nền tảng cho giai đoạn phát triển nên cần phải cực kỳ cẩn thận trong khi tiến hành cuộc họp xét duyệt.

Một khi việc xét duyệt đã hoàn tất thì bản đặc tả các yêu cầu phần mềm sẽ được cả khách hàng lẫn người phát triển "ký tắt". Bản đặc tả trở thành "hợp

đồng" cho việc phát triển phần mềm. Những thay đổi trong yêu cầu được nêu ra sau khi bản đặc tả đã hoàn thành sẽ không bị loại bỏ. Nhưng khách hàng phải lưu ý rằng từng thay đổi sau khi ký đều là một mở rộng của phạm vi phần mềm và do đó có thể làm tăng thêm chi phí và/hoặc kéo dài lịch biểu.

Bản đặc tả rất khó "kiểm thử" theo mọi cách có nghĩa, do đó sự không nhất quán hay bỏ sót có thể bị bỏ qua không chú ý tới. Trong khi xét duyệt, người ta có thể khuyến cáo những thay đổi cho bản đặc tả. Có thể sẽ cực kỳ khó khăn để định lượng tác động toàn cục của thay đổi, tức là làm sao việc thay đổi trong một chức năng lại ảnh hưởng tới các yêu cầu cho chức năng khác? Người ta đã phát triển các công cụ đặc tả tự động hoá để giúp giải quyết vấn đề này.

3.3. Tài liệu đặc tả yêu cầu

Tài liệu yêu cầu mức hệ thống (system requirements) nêu ra các dịch vụ của hệ thống và chi tiết các ràng buộc của nó. Tài liệu này đôi khi còn gọi là *đặc tả chức năng hệ thống*, cần phải rõ ràng và chính xác. Nó được sử dụng làm *cơ sở cho hợp đồng* giữa người mua và người phát triển phần mềm [16].

Đặc tả phần mềm (software specification) là sự mô tả khái quát các chức năng phần mềm trợ giúp hoạt động nghiệp vụ làm cơ sở để thiết kế và triển khai phần mềm sau này. *Tài liệu đặc tả phần mềm* được bổ sung thêm các đặc tả về các phần tử khác trong hệ thống để trở thành *tài liệu đặc tả yêu cầu hệ thống*.

Yêu cầu hệ thống được mô tả theo các mức khác nhau phục vụ cho những người đọc khác nhau. *Yêu cầu người dùng* phải hướng tới mức quản lý, tức là cả khách hàng và người quản lý thầu đều hiểu được. Trong khi đó *đặc tả yêu cầu hệ thống* lại hướng tới nhóm kỹ thuật và những người quản lý dự án. Người sử dụng hệ thống có thể đọc cả hai loại tài liệu nói trên. Cuối cùng, *đặc tả yêu cầu phần mềm* là tài liệu hướng tới việc triển khai. Nó được viết cho các kỹ sư phần mềm, những người tham gia vào tiến trình phát triển hệ thống.

Về nguyên tắc, các yêu cầu của hệ thống phải là vừa đầy đủ, vừa không gây ra mâu thuẫn. Thực tế đối với các hệ lớn và phức tạp thì thực khó đạt được tính đầy đủ và tính không gây mâu thuẫn cho phiên bản đầu của tư liệu yêu cầu phần mềm. Vấn đề là khi duyệt lại hoặc trong các pha sau này của vòng đời phần mềm người ta phát hiện ra các sự không thoả mãn đó thì tư liệu yêu cầu phải được chỉnh lý lại.

Việc xác định yêu cầu là việc đặc tả hướng khách hàng và được *viết bởi ngôn ngữ của khách hàng*. Khi đó có thể dùng các khái niệm không chính xác, ngôn ngữ tự nhiên và các biểu đồ. *Đặc tả yêu cầu* (được gọi là *đặc tả chức năng*) *là cơ sở cho hợp đồng làm hệ thống:* nó phải không được mơ hồ, mà mơ hồ đó có thể dẫn tới sự hiểu lầm bởi khách hàng hoặc bởi người ký hợp đồng. Đặc tả yêu cầu thường được trình bày bằng các mô hình hệ thống.

Rất nhiều vấn đề của kỹ nghệ phần mềm là khó khăn đối với đặc tả yêu cầu. Với một yêu cầu mơ hồ thì người phát triển sẽ thực hiện nó sao cho rẻ nhất. Trong khi đó khách hàng lại không muốn như vậy. Sau các cuộc đấu khẩu kéo dài thì các yêu cầu mới sẽ được thiết lập và có các đổi thay đối với hệ thống. Dĩ nhiên việc đó làm trì hoãn ngày phân phát hệ thống và làm tăng chi phí.

Chi phí do các sai sót trong việc phát biểu các yêu cầu có thể là rất cao, đặc biệt là nếu các sai này còn vẫn chưa được phát hiện trước khi hệ thống được thực hiện. Boehm báo cáo rằng trong một vài hệ thống lớn thì có đến 95% các mã đã phải viết lại để thoả mãn các yêu cầu của người dùng đã đổi thay và 12% các lỗi được phát hiện trong khoảng ba năm đầu sử dụng là các lỗi do đặc tả yêu cầu không đúng đắn. Chi phí do thay đổi một yêu cầu là đắt hơn nhiều so với chi phí sửa chữa thiết kế hoặc do lỗi mã.

Khi đặc tả yêu cầu thường phải sử dụng những công cụ đặc biệt, như các mô hình hay các ngôn ngữ đặc tả chuyên dụng vì việc mô tả bằng ngôn ngữ tự nhiên có nhiều hạn chế [9].

Có người đề nghị rằng các ngôn ngữ đặc tả hình thức toán học nên được dùng để biểu thị các yêu cầu hệ thống. Nhưng đa số các khách hàng lại không

hiểu đặc tả hình thức toán học. Hall [23] đề xuất con đường để đi vòng qua khó khăn đó là viết thêm các chú giải dài dòng ngay bên cạnh cho người dùng dễ hiểu. Khi nào người dùng trở nên quen thuộc hơn với những ưu điểm của đặc tả hình thức và các kỹ sư phần mềm không còn miễn cưỡng dùng các phương pháp hình thức thì sẽ có nhiều các đặc tả yêu cầu sẽ dựa trên các mô hình hình thức của chức năng hệ thống.

Hiện tại, có nhiều lựa chọn khác đã được nghiên cứu thay cho việc đặc tả bằng ngôn ngữ tự nhiên, đó là:

- *Ngôn ngữ tự nhiên có cấu trúc;*
- *Ngôn ngữ mô tả chương trình;*
- *Ngôn ngữ đặc tả yêu cầu;*
- *Đặc tả dựa trên mô hình với ký pháp đồ họa;*
- *Đặc tả bằng toán học - hình thức hóa.*

Tuỳ theo các tổ chức cụ thể, đặc tả yêu cầu có thể được thể hiện bằng các cách khác nhau kể từ mức phát biểu bằng ngôn ngữ tự nhiên về các dịch vụ mà hệ cần cung cấp đến mức đặc tả hệ thống một cách hình thức kiểu toán học. Danh giới giữa các yêu cầu và đặc tả hình thức là một thứ rất tinh tế và các thuật ngữ này lại còn có thể được dùng theo nghĩa như nhau.

Bản đặc tả hệ thống là một tài liệu nền tảng cho kỹ nghệ phần cứng, kỹ nghệ phần mềm, kỹ nghệ CSDL và kỹ nghệ con người. Nó mô tả về chức năng và hiệu suất của hệ thống và những ràng buộc hệ thống, quy định cả giới hạn cho từng phần tử hệ thống, mô tả thông tin (dữ liệu và điều khiển) vào và ra khỏi hệ thống.

Dàn bài về bản đặc tả hệ thống sẽ được trình bày trong bảng 3.3. Tuy nhiên cần lưu ý rằng đây chỉ là một trong nhiều dàn bài có thể được dùng để định dạng một tài liệu mô tả hệ thống. Định dạng và nội dung thực tế có thể còn tuỳ vào các chuẩn kỹ nghệ hệ thống hay phần mềm. Nó được điều chỉnh theo yêu cầu và mức ưa chuộng của người dùng.

Bảng 3.3. Dàn bài đặc tả hệ thống [23]:

I. Giới thiệu	A. Phạm vi và mục tiêu của dự án B. Tổng quan 1. Mục tiêu 2. Ràng buộc
II. Mô tả chức năng và dữ liệu	A. Kiến trúc hệ thống 1. Biểu đồ ngữ cảnh kiến trúc 2. Mô tả về biểu đồ ngữ cảnh hệ thống B. Cấu trúc dữ liệu
III. Mô tả các hệ thống con	A. Đặc tả biểu đồ kiến trúc cho hệ con **n** 1. Biểu đồ luồng kiến trúc 2. Chú giải mô-đun hệ thống 3. Vấn đề về hiệu suất 4. Ràng buộc thiết kế 5. Cách tạo các thành phần hệ thống B. Từ điển kiến trúc C. Biểu đồ và mô tả liên nối hệ thống
IV. Các kết quả mô hình hoá và mô phỏng hệ thống	A. Mô hình hệ thống được dùng cho mô phỏng B. Kết quả mô phỏng C. Vấn đề hiệu suất đặc biệt
V. Vấn đề dự án	A. Chi phí xây dựng dự phòng B. Lịch biểu dự phòng
VI. Phụ lục	

Đặc tả hệ thống phải được xét duyệt. Cuộc xét duyệt đặc tả hệ thống nhằm đánh giá tính đúng đắn của định nghĩa được chứa trong bản đặc tả hệ

thống. Mức độ chi tiết trong giai đoạn đánh giá kỹ thuật (họp xét duyệt hệ thống) phụ thuộc vào mức độ chi tiết trong nhiệm vụ xác định ban đầu.

Bản đặc tả các yêu cầu phần mềm được tạo ra tại đỉnh cao của nhiệm vụ phân tích.

Chức năng và hiệu năng được cấp phát cho phần mềm xem như một phần của kỹ nghệ hệ thống thường được làm mịn bằng cách thiết lập phần mô tả thông tin đầy đủ như *mô tả chức năng chi tiết, chỉ báo về các yêu cầu hiệu năng và những ràng buộc thiết kế, các tiêu chuẩn hợp lệ thích hợp và những dữ liệu khác thường cần tới*. Cơ quan tiêu chuẩn quốc gia, IEEE và Bộ quốc phòng Mỹ đề nghị các định dạng cho các đặc tả yêu cầu phần mềm.

Bảng 3.4. Dàn bài đặc tả yêu cầu phần mềm

Dàn bài đặc tả yêu cầu phần mềm	
I. Giới thiệu	
A. Đại cương về hệ thống	(mục tiêu, mục đích)
B. Mô tả chung	(cấu trúc)
C. Các ràng buộc dự án phần mềm	(phạm vi) (chi tiết)
II. Mô tả thông tin	(dựa vào cấu trúc thông tin)
A. Biểu diễn luồng thông tin	(đặc biệt chú ý đến khi xét các hệ thời gian thực)
1. Luồng dữ liệu	
2. Luồng điều khiển	
B. Biểu diễn nội dung thông tin	(cho từng chức năng)
C. Mô tả giao diện hệ thống	(lời giải thích)
III. Mô tả chức năng	
A. Phân hoạch chức năng	(về các mặt, kinh tế, xã hội, kỹ thuật, chi phí, thời gian)
B. Mô tả chức năng	

1. Tường thuật về cách xử lý	(theo từng mô-đun) (có luận giải)
2. Hạn chế/giới hạn	(cấu trúc tổng thể, tương quan với các phần từ hệ thống khác)
3. Yêu cầu hiệu năng	
4. Ràng buộc thiết kế	
5. Biểu đồ trợ giúp	(có luận giải) (xem xét vận hành của phần mềm)
C. Mô tả điều khiển	(mức toàn cảnh, mức đỉnh)
1. Đặc tả điều khiển	(mức cụ thể, bộ phận)
2. Ràng buộc thiết kế	(quan trọng nhất nhưng hay bị bỏ quên)
IV. Mô tả hành vi	
A. Trạng thái hệ thống	
B. Sự kiện và hành động	(tiêu chuẩn hướng tới (lý tưởng))
V. Tiêu chuẩn hợp lệ	
A. Giới hạn hiệu năng	(tài liệu kỹ nghệ phần mềm khác, tham khảo kỹ thuật ,...)
B. Lớp các kiểm thử	
C. Đáp ứng phần mềm dự kiến	(Danh sách các nhà cung cấp, các chuẩn,...)
D. Các xem xét đặc biệt	
VI. Sách tham khảo	
VII. Phụ lục	

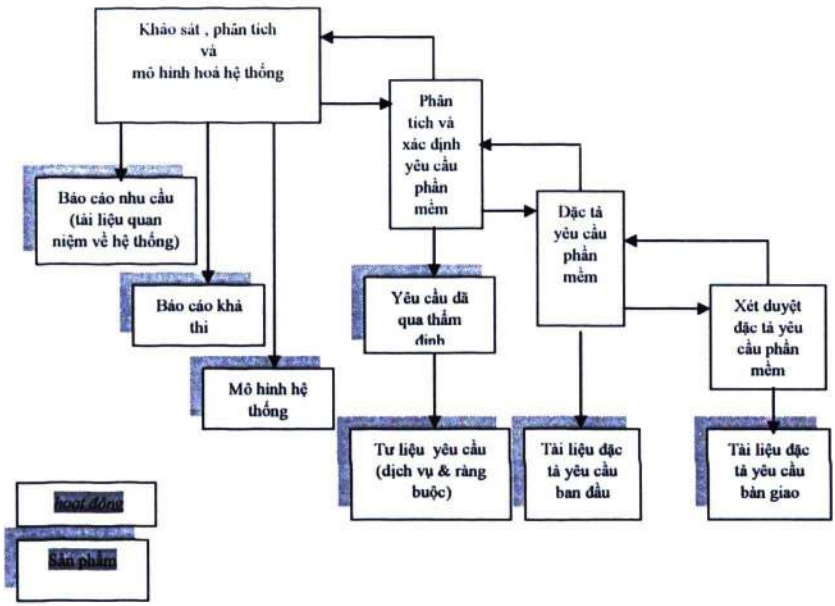
Trong nhiều trường hợp bản *đặc tả các yêu cầu Phần mềm* còn có thể còn có kèm theo một bản mẫu thực hiện được (mà trong một số trường hợp có thể thay thế được cho bản đặc tả), một bản mẫu trên giấy, hay *tài liệu sơ bộ của người dùng*.

Bản *tài liệu sơ bộ của người* dùng trình bày phần mềm như hộp đen. Tức là, chủ yếu nhấn mạnh vào cái vào của người dùng và cái ra kết quả. Tài liệu này có thể dùng như một công cụ có giá trị để làm lộ ra vấn đề ở giao diện người - máy.

3.4. Tiến trình xác định yêu cầu và phát triển bản mẫu

3.4.1. Tiến trình xác định và đặc tả yêu cầu phần mềm

Phân tích và định rõ yêu cầu *là bước kỹ thuật đầu tiên* trong tiến trình kỹ nghệ phần mềm. Hoạt động *phân tích và định rõ yêu cầu* hướng tới đặc tả yêu cầu phần mềm được thể hiện trong tiến trình sau:



Hình 3.1. Tiến trình phân tích và định rõ yêu cầu phần mềm

Phân tích là công việc trung tâm khi xây dựng một hệ thống. Nó gồm nhiều giai đoạn và khởi sự ngay trong giai đoạn lập kế hoạch vì công việc tìm hiểu nhu cầu và khả năng thực tế đã bắt đầu ngay từ khi làm chiến lược. Việc phân tích được chia thành nhiều công đoạn cụ thể.

Phân tích hệ thống là việc sử dụng các phương pháp và công cụ để nhận thức và hiểu biết được hệ thống, tìm các giải pháp giải quyết các vấn đề phức tạp đặt ra cho hệ thống được nghiên cứu.

Phân tích hệ thống nhằm xác định nhu cầu thông tin của tổ chức. Nó sẽ cung cấp những dữ liệu cơ sở cho việc thiết kế hệ thống sau này. Phân tích chi tiết bao gồm phân tích dữ liệu và phân tích các hoạt động xử lý mà hệ thống tương lai sẽ sử dụng và thực hiện.

Trước khi phân tích cần tiến hành khảo sát những bộ phận của tổ chức thuộc phạm vi liên quan đến dự án. Những dữ liệu thu được phục vụ cho việc xây dựng mô hình xử lý của hệ thống cùng các tài liệu bổ sung khác.

Giai đoạn *khảo sát, nghiên cứu hiện trạng* nhằm hiểu rõ tình trạng hoạt động của hệ thống cũ trong mục đích hoạt động của tổ chức. Nó bao gồm việc nghiên cứu hoạt động của tổ chức thông qua xem xét nhiệm vụ của tổ chức, vai trò và nhiệm vụ của các tổ chức thành viên cũng như các tổ chức bên ngoài có liên quan, mối quan hệ thông tin (thông tin vào, thông tin ra, luồng thông tin, vai trò trong hoạt động xử lý thông tin để phục vụ hoạt động của hệ thống, mật độ, tần số, thời hạn...).

Nội dung phân tích hệ thống bao gồm:

- Phân tích mục tiêu: xác định một cách chính xác và đầy đủ mục tiêu mà hệ thống và cũng là mục tiêu mà chủ thể mong muốn. Trong thực tế, mục tiêu của hệ thống không phải luôn rõ ràng.

- Phân tích phạm vi và môi trường của hệ thống: xác định phạm vi hệ thống cần nghiên cứu liên quan đến việc đạt được mục tiêu ở trên.

- Phân tích cấu trúc: xác định các phần tử, các mối quan hệ của hệ thống. Đó là những đặc trưng quan trọng của hệ thống.

- Phân tích các nguồn lực và tác động của nó: Cần có các nguồn lực cân để đạt mục tiêu. Đó là những nguồn lực nào? Khả năng khai thác? Quy mô và mức độ tác động của nó lên hệ thống?

- Phân tích tổng hợp toàn hệ thống: để thấy được những đặc tính trồi có thể của hệ thống khi nó vận hành.

Việc phân tích và định rõ nhu cầu là giai đoạn đầu của tiến trình thiết lập các *dịch vụ (*mà hệ thống phải giải quyết) và *các ràng buộc (*mà hệ thống phải tuân theo). Kết quả của công việc này là bản *đặc tả yêu cầu (chức năng, hệ thống, phần mềm)*. Đó thường là *tư liệu chính thức* đầu tiên được tạo ra trong tiến trình phần mềm.

Khó khăn có tính nguyên lý của công việc thiết kế các hệ thống phần mềm là các vấn đề *của độc*, nó là các vấn đề không có công thức định nghĩa. Mỗi phát biểu hình thức (các yêu cầu) của vấn đề là rất riêng và sự nhận biết của người phát triển trong tiến trình là liên tục thay đổi.

3.4.2. Tiến trình phát triển bản mẫu hệ thống

Việc tạo bản mẫu có một ý nghĩa lớn vì duyệt lại là một phần của tiến trình thẩm định yêu cầu. Từ đặc tả yêu cầu tưởng tượng được hệ thống sẽ được sử dụng như thế nào. Một chức năng được mô tả trong một đặc tả là hữu ích và đã được xác định tốt nhưng thực tế việc sử dụng chức năng đó kết hợp với các chức năng khác lại để lộ ra rằng cái nhìn ban đầu của NSD đã không đúng và không đầy đủ.

Một cách giải quyết khó khăn đó là phát triển một *bản mẫu hệ thống* cho phép NSD thí nghiệm trên bản mẫu đó. Các yêu cầu của hệ thống được thẩm định và người dùng phát hiện ra các sai sót.

Bản mẫu phần mềm hoàn toàn khác với bản mẫu phần cứng. Khi phát triển các hệ thống phần cứng, thì thực tế người ta phát triển một bản mẫu hệ thống để thẩm định thiết kế hệ thống. Một bản mẫu hệ thống điện tử có thể được thực hiện và được thử nghiệm bằng cách dùng các *thành phần chưa được lắp ráp vào vỏ* trước khi đầu tư vào các mạch tích hợp chuyên dụng đắt tiền để thực hiện một đời sản phẩm mới của hệ thống. *Một bản mẫu phần mềm* không phải nhằm vào việc thẩm định thiết kế mà là để thẩm định yêu cầu của NSD.

Lợi ích của việc phát triển bản mẫu có thể kể ra gồm:

- *Sự hiểu lầm* giữa những người phát triển phần mềm và những người sử dụng phần mềm có thể được nhận thấy rõ khi các chức năng của hệ thống được trình diễn

- *Sự thiếu hụt các dịch vụ người dùng* có thể được phát hiện

- *Sự khó sử dụng hoặc sự nhầm lẫn các dịch vụ người dùng* có thể được thấy rõ và được sửa lại

- Đội ngũ phát triển phần mềm *có thể tìm ra được* các yêu cầu không đầy đủ hoặc không kiên định khi họ phát triển bản mẫu

- Một hệ thống hoạt động được (mặc dầu là hạn chế) là bằng chứng *thuyết minh cho tính khả thi và tính hữu ích* của ứng dụng đó cho các nhà quản lý

- Bản mẫu đó được dùng làm cơ sở cho *việc viết đặc tả một sản phẩm*

Mặc dù mục tiêu chủ yếu của việc tạo bản mẫu là để thẩm định các yêu cầu phần mềm cũng nên chỉ ra rằng một bản mẫu phần mềm cũng có các ứng dụng khác như:

+ Dùng để *huấn luyện NSD* ngay từ trước khi hệ thống được phân phối;

+ Dùng *trong tiến trình thử nghiệm hệ thống*. Điều đó nghĩa là cùng các trường hợp thử như nhau vừa dùng cho thử bản mẫu vừa cho thử hệ thống. Kết quả khác nhau có nghĩa là có sai sót.

Tạo bản mẫu là một *kỹ thuật giảm bớt rủi ro*. Một rủi ro lớn trong việc phát triển phần mềm là các sai sót mà đến giai đoạn cuối mới phát hiện và chỉnh sửa là rất tốn kém. Kinh nghiệm cho hay rằng việc tạo bản mẫu sẽ giảm bớt số các vấn đề của đặc tả yêu cầu và giá cả tổng cộng của việc phát triển có thể là thấp hơn nếu ta phát triển bản mẫu.

Tiến trình phát triển bản mẫu hệ thống được triển khai gồm 4 bước:

- Bước 1-Thiết lập các mục tiêu *của việc tạo bản mẫu, quyết định chọn input, output cho bản mẫu hệ thống;*

- Bước 2-Chọn các chức năng *cho việc tạo bản mẫu và quyết định những đặc tả phi chức năng nào cần phải tạo bản mẫu hệ thống;*

- Bước 3-Phát triển *hệ bản mẫu phần mềm;*

- Bước 4-Đánh giá *hệ bản mẫu phần mềm.*

Cần phải làm rõ ràng các mục tiêu tạo bản mẫu trước khi bắt đầu công việc. Mục tiêu đó có thể là phát triển một hệ thống liên quan chủ yếu đến tạo bản mẫu giao diện người dùng; nó cũng có thể là việc phát triển một hệ thống thẩm

định các yêu cầu hệ thống chức năng, nó cũng có thể là phát triển một hệ thống để trình diễn tính khả thi của ứng dụng cho các nhà quản lý xem xét,... Cùng một bản mẫu không thể đạt được tất cả các mục tiêu. Nếu mục tiêu còn chưa rõ ràng thì người quản lý hoặc người sử dụng có thể hiểu nhầm chức năng của nguyên mẫu và không đạt được lợi ích thiết thực đầy đủ của việc tạo bản mẫu.

Giai đoạn tiếp theo trong tiến trình này là quyết định xem cái gì được đưa vào và cái gì được lấy ra từ hệ bản mẫu đó. Bản mẫu của phần mềm là đắt nếu nó được thực hiện bằng cách dùng cùng các công cụ và các chuẩn y như cho chính hệ thống cuối cùng.

Bởi vậy, có thể quyết định tạo bản mẫu tất cả các chức năng của hệ thống nhưng ở mức thu gọn. Cũng có thể chỉ một số chức năng hệ thống được tạo mẫu. Bình thường trong thực tế người ta hay bỏ bớt các yêu cầu phi chức năng chẳng hạn như yêu cầu về tốc độ, về không gian nhớ. Việc xử lý sai sót và việc quản lý có thể lờ đi và có thể là dư thừa trừ phi mục tiêu của việc tạo mẫu là thiết lập một giao diện người máy. Các chuẩn của độ tin cậy và của chất lượng chương trình có thể chưa tính đến.

Giai đoạn cuối cùng của tiến trình này là đánh giá bản mẫu, có người cho rằng đây là giai đoạn quan trọng nhất của việc tạo bản mẫu. Phải dự tính cho việc huấn luyện người sử dụng trong giai đoạn này và và các mục tiêu của việc tạo mẫu hẳn là nên dẫn ra một kế hoạch đánh giá. Phải có đủ thời gian cho NSD trở nên dễ chịu với hệ thống và thiết lập một mẫu sử dụng chuẩn tắc và bởi vậy phát hiện được các sai sót yêu cầu.

Việc tạo bản mẫu là kỹ thuật chính trong mô hình tiến trình xoắn ốc hỗ trợ định giá rủi ro.

3.4.3. Tiến trình phát triển bản mẫu phần mềm

Vấn đề cơ bản là khó đánh giá phần mềm mới được xây dựng có ảnh hưởng thế nào tới công việc của người dùng. Đối với các hệ thống mới, đặc biệt là đối với các hệ thống lớn và phức tạp thì có lẽ là không thể đánh giá được trước khi hệ thống được xây dựng xong và đưa vào ứng dụng

Một cách để vượt qua khó khăn này là *sử dụng cách tiếp cận thăm dò* để phát triển hệ thống. Điều này có nghĩa là trình bày cho người dùng một hệ

thống chưa đầy đủ và rồi cải biên nó tăng cường hệ thống đó khi mà các yêu cầu thực của người dùng trở nên trong suốt. Sau khi đánh giá bản mẫu đó bị thải loại và một hệ chất lượng tốt hơn được xây dựng.

Tiến trình làm bản mẫu phần mềm gồm 6 bước:

Bước 1: Đánh giá yêu cầu phần mềm và xác định liệu phần mềm cần xây dựng có xứng đáng để làm bản mẫu không.

Không phải tất cả các phần mềm đều có thể đưa tới làm bản mẫu. Ta có thể xác định một số nhân tố làm bản mẫu: lĩnh vực ứng dụng, độ phức tạp ứng dụng, đặc trưng khách hàng và đặc trưng dự án.

Nói chung, bất kỳ ứng dụng nào tạo ra việc hiển thị trực quan động, tương tác nhiều với con người, hay yêu cầu các thuật toán hay việc xử lý tổ hợp mà cần phải được xây dựng theo một cách tiến hoá thì đều có thể làm bản mẫu. Tuy nhiên, những lĩnh vực ứng dụng này phải được cân nhắc dựa trên độ phức tạp của ứng dụng. Nếu một ứng dụng yêu cầu xây dựng tới 10 ngàn dòng mã lệnh thì phần chắc là nó quá phức tạp không thể làm bản mẫu được. Tuy nhiên nếu ta có thể phân hoạch độ phức tạp thì vẫn có thể làm bản mẫu cho từng phần của phần mềm.

Để bảo đảm tính tương tác giữa khách hàng với bản mẫu cần xét đến 2 yếu tố:

1- Khách hàng phải cam kết dùng tài nguyên để đánh giá và làm mịn bản mẫu;

2- Khách hàng phải có khả năng đưa ra những quyết định về yêu cầu một cách kịp thời.

Bản chất của dự án quyết định tính hiệu quả của làm bản mẫu.

Bước 2: Với một dự án chấp thuận được, người phân tích biểu diễn vắn tắt yêu cầu.

Trước khi có thể bắt đầu xây dựng một bản mẫu, người phân tích phải biểu diễn miền thông tin và các lĩnh vực hành vi và chức năng của vấn đề rồi xây dựng một cách tiếp cận hợp lý tới việc phân hoạch. Có thể ứng dụng các nguyên lý phân tích nền tảng (trên xuống) và các phương pháp phân tích yêu cầu.

Bước 3: Sau khi đã duyệt xét mô hình yêu cầu, tạo ra một đặc tả thiết kế vắn tắt cho bản mẫu.

Việc thiết kế phải xuất hiện trước khi bắt đầu làm bản mẫu. Tuy nhiên thiết kế tập trung chủ yếu vào các vấn đề thiết kế dữ liệu và kiến trúc mức đỉnh chứ không tập trung vào thiết kế thủ tục chi tiết.

Bước 4: Phần mềm bản mẫu được tạo ra, kiểm thử và làm mịn.

Một cách lý tưởng, các khối xây dựng phần mềm hiện có được dùng để tạo ra bản mẫu một cách nhanh chóng.

Bước 5: Một khi đã kiểm thử xong bản mẫu thì có thể trình bày nó cho khách hàng.

Việc kiểm thử ứng dụng và gợi ý những thay đổi là cốt lõi của cách tiếp cận làm bản mẫu. Chính ở đây mà khách hàng có thể xem xét cách biểu diễn được cài đặt cho yêu cầu phần mềm, gợi ý những thay đổi làm cho phần mềm đáp ứng tốt hơn với các nhu cầu thực tế.

Bước 6: Lặp lại các bước 4 và 5 cho tới khi tất cả các yêu cầu đã được hình thức hoá hay cho tới khi bản mẫu đã tiến hoá thành một hệ thống sản xuất.

TÓM TẮT CHƯƠNG

Yêu cầu cho một hệ thống mô tả những việc mà hệ thống làm và những ràng buộc mà nó phải tuân thủ khi hoạt động. Yêu cầu hệ thống bao gồm yêu cầu chức năng, yêu cầu phi chức năng, yêu cầu miền lĩnh vực, yêu cầu người dùng.

Có hai phương pháp khác nhau để tiếp cận việc xác định yêu cầu. Đó là Phương pháp mô hình và phương pháp hướng cách nhìn.

Có ba kỹ thuật phân tích yêu cầu điển hình là kỹ thuật phân tích yêu cầu dựa trên phương pháp mô hình, kỹ thuật VORD và kỹ thuật phân tích hình thức hoá.

Để phát hiện được các yêu cầu, cả hai cách tiếp cận hướng chức năng và hướng đối tượng đều tập trung hướng vào mô tả mặt nghiệp vụ của hệ thống

thực và kết quả thu được là một mô hình nghiệp vụ. Mô hình nghiệp vụ được xây dựng trên cơ sở của phương pháp phân tích hệ thống bao gồm: Một mô hình ngữ cảnh; Các mô hình cấu trúc chức năng; Mô tả chi tiết các chức năng; Mô tả các đối tượng dữ liệu; Mô tả mối liên kết giữa dữ liệu và chức năng; Từ điển giải thích.

Yêu cầu phần mềm là các đòi hỏi không thể thiếu về các chức năng (dịch vụ) mà phần mềm dự kiến phải cung cấp và các ràng buộc mà phần mềm và cả hệ thống phải tuân thủ khi vận hành.

Bản đặc tả các yêu cầu phần mềm được tạo ra tại đỉnh cao của nhiệm vụ phân tích. Bản đặc tả trở thành "hợp đồng" cho việc phát triển phần mềm. Còn bản đặc tả hệ thống là một tài liệu nền tảng cho kỹ nghệ phần cứng, kỹ nghệ phần mềm, kỹ nghệ CSDL và kỹ nghệ con người.

Tài liệu yêu cầu mức hệ thống nêu ra các dịch vụ của hệ thống và chi tiết các ràng buộc của nó. Tài liệu này đôi khi còn gọi là đặc tả chức năng hệ thống, cần phải rõ ràng và chính xác. Nó được sử dụng làm cơ sở cho hợp đồng giữa người mua và người phát triển phần mềm

Phân tích và định rõ yêu cầu là bước kỹ thuật đầu tiên trong tiến trình kỹ nghệ phần mềm. Phân tích là công việc trung tâm khi xây dựng một hệ thống. Nó gồm nhiều giai đoạn và khởi sự ngay trong giai đoạn lập kế hoạch vì công việc tìm hiểu nhu cầu và khả năng thực tế đã bắt đầu ngay từ khi làm chiến lược. Việc phân tích được chia thành nhiều công đoạn cụ thể. Phân tích hệ thống nhằm xác định nhu cầu thông tin của tổ chức. Nó sẽ cung cấp những dữ liệu cơ sở cho việc thiết kế hệ thống sau này. Phân tích chi tiết bao gồm phân tích dữ liệu và phân tích các hoạt động xử lý mà hệ thống tương lai sẽ sử dụng và thực hiện.

Việc tạo bản mẫu có một ý nghĩa lớn vì duyệt lại là một phần của tiến trình trình thẩm định yêu cầu. Bản mẫu phần mềm hoàn toàn khác với bản mẫu phần cứng. Tạo bản mẫu là một kỹ thuật giảm bớt rủi ro.

Tiến trình làm bản mẫu phần mềm gồm 6 bước: 1-Đánh giá yêu cầu phần mềm và xác định liệu phần mềm cần xây dựng có xứng đáng để làm bản mẫu

không, 2-Biểu diễn vắn tắt yêu cầu, 3-Tạo ra một đặc tả thiết kế vắn tắt cho bản mẫu, 4-Kiểm thử và làm mịn phần mềm bản mẫu được tạo ra, 5-Trình bày bản mẫu cho khách hàng, 6-Lặp lại các bước 4 và 5 cho tới khi tất cả các yêu cầu đã được hình thức hoá hay cho tới khi bản mẫu đã tiến hoá thành một hệ thống sản xuất.

CÂU HỎI CHƯƠNG 3

3.1. Đặc tả khác mô tả ở chỗ nào?

3.2. Đặc tả bằng ngôn ngữ tự nhiên có những hạn chế nào?

3.3. Tại sao bản Đặc tả hệ thống cần phải được xét duyệt?

3.4. Tại sao nói Bản đặc tả các yêu cầu phần mềm được tạo ra tại đỉnh cao của nhiệm vụ phân tích?

3.5. Phân tích hệ thống là gì?

3.6.Tại sao nói Phân tích là công việc trung tâm khi xây dựng một hệ thống?

3.7. Phân tích hệ thống nhằm mục đích gì?

3.8. Bản mẫu phần mềm hoàn toàn khác với bản mẫu phần cứng?

3.9. Tại sao nói rằng việc tạo bản mẫu là một *kỹ thuật giảm bớt rủi ro?*

3.10. Ý tưởng của *cách tiếp cận thăm dò* để phát triển hệ thống là gì?

BÀI TẬP CHƯƠNG 3

Bài tập 3.1:

Có một hệ thống kinh doanh bán buôn được mô tả trong bài tập 1.1 chương 1.

Yêu cầu:

1- *Vẽ mô hình ngữ cảnh theo hướng cấu trúc cho hệ thống?*

3- Mô tả ma trận liên kết giữa chức năng và dữ liệu?

Bài tập 3.2:

Hệ thống rút tiền tự động ATM được mô tả như sau:

"Giả sử có các máy rút tiền tự động ATM đặt ở những vị trí khác nhau (tại các phố, khu công sở, trung tâm thương mại) trong thành phố. Chúng được nối với Trung tâm tại trụ sở Ngân hàng thông qua hệ thống mạng máy tính. Máy tính trung tâm lưu trữ và quản trị CSDL khách hàng, xử lý những công việc chuyên ngành của ngân hàng và yêu cầu ATM trả tiền.

Máy rút tiền tự động bao gồm máy đọc thẻ từ, màn hình và bàn phím để tương tác với người sử dụng.

Khách hàng có thể rút tiền tự động, chuyển tiền, xem số dư trong tài khoản và thực hiện thanh toán với hệ thống tín dụng của Ngân hàng.

Trong những trường hợp đặc biệt như bị mất thẻ (bị lộ số thẻ căn cước), hay muốn thay đổi số thẻ thì khách hàng có thể thay đổi số thẻ căn cước cá nhân (PIN) và tương tự như vậy, khi khách hàng báo bị mất thẻ thì Ngân hàng cũng có thể quyết định thay đổi số thẻ và báo cho khách hàng biết để đảm bảo không cho những người không phải chủ sở hữu rút được tiền".

Yêu cầu:

1- Vẽ biểu đồ ca sử dụng của ATM?

2- Vẽ biểu đồ lớp của ATM?

Chương 4
ĐẶC TẢ THIẾT KẾ

Thiết kế là một quá trình sáng tạo, đòi hỏi kinh nghiệm và sự tinh nhanh của người thiết kế. Thiết kế phải được thực hành và học bằng kinh nghiệm, bằng khảo sát các hệ đang tồn tại ngoài việc học bằng sách vở.

Chương này giới thiệu về đặc tả thiết kế theo các phương diện dữ liệu, cấu trúc chương trình, giao diện và mô-đun xử lý.

Học viên cần nắm vững khái niệm về thiết kế phần mềm, các nội dung thiết kế dữ liệu, cần rèn luyện kỹ năng thiết kế cấu trúc chương trình, thiết kế giao diện cũng như thiết kế mô-đun xử lý.

4.1. Thiết kế và thiết kế phần mềm

Theo Pressman [23]: "*Thiết kế là tiến trình áp dụng nhiều kỹ thuật và nguyên lý với mục đích xác định ra một thiết bị, một tiến trình hay một hệ thống đủ chi tiết để cho phép thực hiện nó về mặt vật lý.*"

Mục tiêu thiết kế là để tạo ra một mô hình hay biểu diễn của một thực thể mà sau này sẽ được xây dựng. Thiết kế cần mô tả sao cho đạt mức có thể làm cơ sở cho thực hiện chi tiết, làm phương tiện liên lạc giữa các nhóm thiết kế các hệ con và cung cấp đủ thông tin cho những người bảo trì hệ thống.

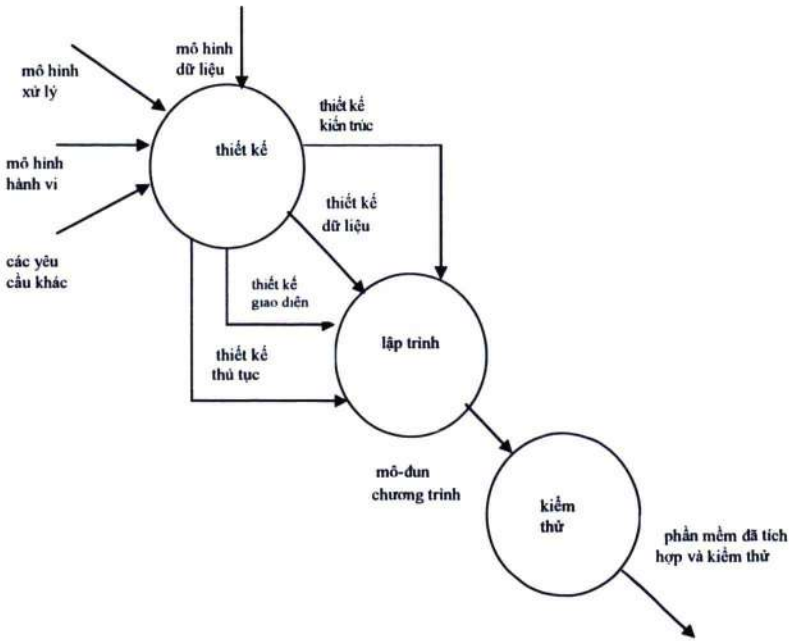
Người ta thường dùng đồ thị, các ngôn ngữ mô tả chương trình, văn bản không hình thức để tạo dựng các tài liệu thiết kế.

Thiết kế phần mềm là một tiến trình chuyển hoá các yêu cầu thành một biểu diễn phần mềm. Thiết kế phần mềm là một quá trình, từ biểu diễn mô tả theo quan điểm tổng thể về phần mềm đến việc làm mịn tiếp sau dẫn tới một biểu diễn thiết kế rất gần với chương trình gốc.

Thiết kế phần mềm là một mô hình của thế giới thực mô tả các thành phần phần mềm và các mối quan hệ của chúng với nhau.

Thiết kế phần mềm nằm ở trung tâm kỹ thuật của tiến trình kỹ nghệ phần mềm và được áp dụng bất kể tới khuôn cảnh phát triển được sử dụng. Một khi các yêu cầu phần mềm đã được phân tích và đặc tả thì thiết kế phần mềm là một trong ba hoạt động kỹ thuật - *thiết kế, lập trình, kiểm thử* - những hoạt động cần để xây dựng và kiểm chứng phần mềm. Từng hoạt động này biến đổi thông tin theo cách cuối cùng tạo ra phần mềm máy tính hợp lệ.

Luồng thông tin trong giai đoạn kỹ thuật này của tiến trình kỹ nghệ phần mềm được minh hoạ trong hình 4.1:



Hình 4.1. Ba hoạt động kỹ thuật trong kỹ nghệ phần mềm [9]

Các yêu cầu phần mềm, được biểu thị bởi các mô hình thông tin, chức năng và hành vi, là cái vào cho bước thiết kế. Bằng việc sử dụng một trong một số các phương pháp thiết kế, bước thiết kế tạo ra thiết kế dữ liệu, thiết kế cấu trúc chương trình, thiết kế thủ tục (mô-đun xử lý) và thiết kế giao diện,...

Thành quả của mỗi hoạt động thiết kế là một đặc tả. Đặc tả này có thể là một đặc tả trừu tượng, hình thức và được tạo ra để làm rõ các yêu cầu, nó cũng

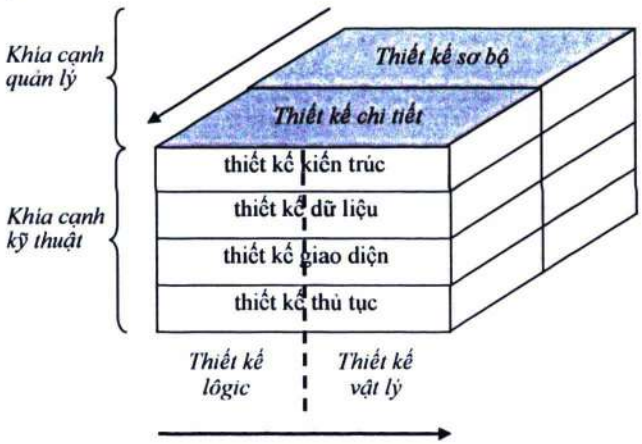
có thể là một đặc tả về một phần nào đó của hệ thống phải được thực hiện như thế nào. Khi quá trình thiết kế tiến triển thì các chi tiết ngày càng được bổ sung vào đặc tả đó. Các kết quả cuối cùng là các đặc tả về các thuật toán và các cấu trúc dữ liệu được dùng làm cơ sở cho việc thực hiện hệ thống.

Theo quan điểm quản lý dự án, thiết kế phần mềm được tiến hành theo hai bước:

1- Thiết kế sơ bộ: quan tâm tới việc chuyển hoá các yêu cầu thành kiến trúc dữ liệu và phần mềm.

2- Thiết kế chi tiết: tập trung vào việc làm mịn biểu diễn kiến trúc để dẫn tới cấu trúc dữ liệu chi tiết và biểu diễn thuật toán cho phần mềm. Thiết kế chi tiết này thường trải qua 2 giai đoạn là thiết kế logic và thiết kế vật lý.

Trong phạm vi thiết kế sơ bộ và chi tiết, có xuất hiện một số hoạt động thiết kế khác nhau. Bên cạnh việc thiết kế dữ liệu, kiến trúc và thủ tục, nhiều ứng dụng hiện đại có hoạt động *thiết kế giao diện* phân biệt. Thiết kế giao diện lập ra cách bố trí và cơ chế tương tác cho tương tác người máy. Mối quan hệ giữa các khía cạnh kỹ thuật và quản lý của thiết kế được minh hoạ trong hình vẽ dưới đây:



Hình 4.2. Quá trình thiết kế nhìn theo các cách khác nhau

4.2. Thiết kế dữ liệu

Trong thiết kế dữ liệu, các cấu trúc dữ liệu được dùng trong việc thực hiện hệ thống sẽ được thiết kế chi tiết và được đặc tả. Thiết kế dữ liệu là việc chuyển mô hình lĩnh vực thông tin đã được tạo ra trong bước phân tích thành các cấu trúc dữ liệu sẽ cần cho việc cài đặt phần mềm.

4.2.1. Thiết kế cơ sở dữ liệu mức logic

Đầu vào của thiết kế CSDL mức logic là mô hình khái niệm dữ liệu. Từ mô hình khái niệm dữ liệu, chúng ta chuyển nó sang mô hình thực thể - mối quan hệ (sơ đồ E_R) theo quy trình rất quen thuộc mức đại học [10]:

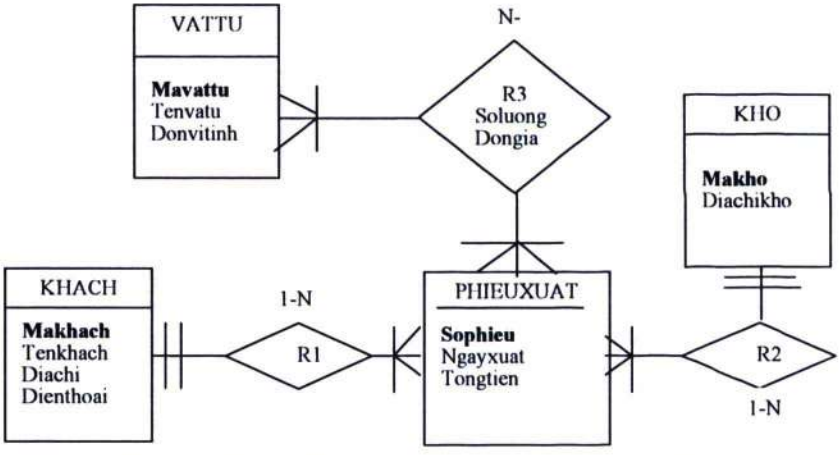
1- Chuyển mô hình khái niệm dữ liệu sang mô hình quan hệ,

2- Chuẩn hoá mô hình quan hệ đến 3NF,

3- Vẽ sơ đồ E_R tương ứng với mô hình quan hệ đã đạt 3NF.

Ví dụ 4.1:

Xuất phát từ mô hình khái niệm sau:



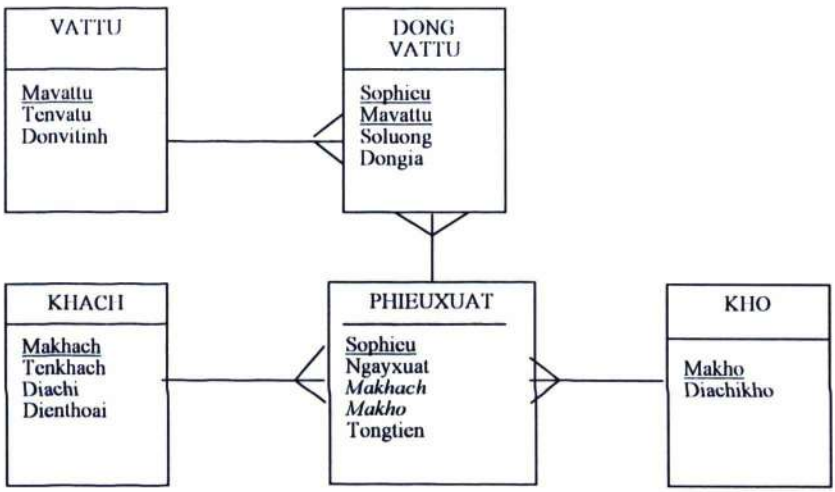
Hình 4.3. Mô hình khái niệm dữ liệu bài toán quản lý vật tư

chúng ta thu được mô hình quan hệ sau:

1. KHACH(Makhach,Tenkhach,Diachi,Dienthoai)
2. KHO(Makho,Diachikho)
3. VATTU (Mavattu,Tenvatu,Donvitinh)
4. PHIEUXUAT(Sophieu,Ngayxuat,*Makhach,Makho,*Tongtien)
5. DONG VATTU(Sophieu,Mavattu,Soluong,Dongia)

Các lược đồ quan hệ trong mô hình quan hệ vừa thu được ở trên đã đạt 3NF.

Cuối cùng, chúng ta vẽ được sơ đồ E_R tương ứng:



Hình 4.4. Mô hình thiết kế CSDL mức logic bài toán quản lý bán vật tư

4.2.2. Thiết kế cơ sở dữ liệu mức vật lý

Mỗi hình chữ nhật trong sơ đồ E_R được chuyển thành một bảng cấu trúc tệp bao gồm các cột chính như tên trường, kiểu dữ liệu, kích cỡ tối đa, khuôn dạng, ràng buộc, mô tả,... Sau khi chuyển hết các hình chữ nhật như vậy, chúng ta sẽ thu được một CSDL cho bài toán.

Ví dụ 4.2

Trong bài toán quản lý bán vật tư, hình chữ nhật mang tên PHIEUXUAT được chuyển thành tệp cấu trúc như sau:

Bảng 4.1. PHIEUXUAT

Tên trường	Kiểu DL	Cỡ DL	Khuôn dạng	Ràng buộc	Diễn giải
so_phieu	ký tự	6	Chữ hoa + số	khoá chính	NOTNULL
makhach	ký tự	15	Chữ hoa + số	khoá ngoại (FK)	Tham chiếu đến bảng KHACH
ngayxuat	ngày tháng	8	dd/mm/yy		Ngày/tháng/năm
makho	ký tự	4	Chữ cái+số	khoá ngoại (FK)	Tham chiếu đến bảng KHO
tongtien	số	15	.##		Dùng 2 số sau chữ số thập phân

4.3. Thiết kế cấu trúc chương trình

Trong thiết kế kiến trúc hệ thống, các hệ con tạo nên hệ tổng thể và các quan hệ của chúng được minh định và ghi thành tài liệu. Đối với mỗi hệ con, một đặc tả trừu tượng các dịch vụ mà nó cung cấp và các ràng buộc phải tuân theo được cung cấp. Trong thiết kế kiến trúc hệ thống, người ta tập trung vào thiết kế cấu trúc chương trình. Thiết kế kiến trúc là định nghĩa ra mối quan hệ giữa các thành phần cấu trúc chính của chương trình.

Sơ đồ luồng dữ liệu chỉ ra cách thức biến đổi dữ liệu vào thành dữ liệu ra thông qua một dãy các phép biến đổi. Đó là một cách hữu ích để mô tả một hệ thống và nó cũng là dễ hiểu không cần một sự huấn luyện đặc biệt nào. Bước thứ nhất của thiết kế xử lý hướng cấu trúc là phát triển một *biểu đồ luồng dữ liệu mức logic* từ biểu đồ luồng dữ liệu mức vật lý trong giai đoạn đặc tả yêu cầu khi bỏ hết các yếu tố vật lý và cấu trúc lại để mô hình vẫn bảo đảm thực hiện các chức năng hệ thống. Sơ đồ này là sơ đồ luồng dữ liệu nghiệp vụ tuy chưa bao gồm các thông tin điều khiển nhưng đã chỉ ra được tư liệu về các phép biến đổi dữ liệu.

Chúng ta xác định *luồng dữ liệu hệ thống* từ sơ đồ luồng dữ liệu nghiệp vụ. Chú ý rằng trong sơ đồ này, sự khác biệt giữa chúng không chỉ là việc phân định rõ tiến trình nào máy làm, tiến trình nào người làm mà còn cả có sự khác biệt về kho dữ liệu nữa. Trong mô hình luồng dữ liệu hệ thống chỉ xuất hiện các kho dữ liệu ứng với các bảng cấu trúc tệp trong thiết kế CSDL vật lý, chứ không phải là các hồ sơ dữ liệu vào xác định ngay từ giai đoạn khảo sát và phân tích, nghĩa là diện mạo mô hình có khác đôi chút do phải chỉnh lại các luồng dữ liệu vào ra kho dữ liệu.

Từ sơ đồ luồng dữ liệu hệ thống, xây dựng được *lược đồ cấu trúc chương trình* trong thiết kế chương trình. Đó vừa là thiết kế kiến trúc, vừa là thiết kế xử lý. Lược đồ cấu trúc chỉ ra cấu trúc các thành phần theo thứ bậc của hệ thống. Nó chỉ ra rằng các phần tử của một sơ đồ luồng dữ liệu có thể được thực hiện như thế nào với tư cách là một thứ bậc của các đơn vị chương trình. Lược đồ cấu trúc có thể được dùng như là một mô tả chương trình nhìn thấy được với các thông tin xác định các sự lựa chọn và các vòng lặp. Lược đồ cấu trúc được dùng chỉ để trình bày một tổ chức tĩnh của thiết kế.

4.4. Thiết kế giao diện

4.4.1. Ý nghĩa và bản chất của thiết kế giao diện người - máy

Thiết kế hệ thống máy tính bao gồm một loạt hoạt động từ thiết kế phần cứng tới thiết kế giao diện. Các kỹ sư phần mềm thường phải chịu trách nhiệm về thiết kế giao diện cũng như thiết kế các thành phần phần mềm khác.

Thiết kế giao diện là thể hiện ý tưởng chuyển các chức năng được mô tả bằng sơ đồ luồng hệ thống sang dạng một màn hình thao tác để qua đó con người tác động nhằm thực hiện các chức năng đó trên máy.

Giao diện người - máy là cơ chế qua đó thiết lập đối thoại giữa chương trình và người dùng. Vì vậy, khi nhân tố con người được tính tới thì đối thoại sẽ trôi chảy và hoà điệu giữa người dùng và chương trình sẽ được thiết lập. Nếu nhân tố này bị bỏ qua thì hệ thống bị coi là "không thân thiện".

Tính dễ sử dụng giao diện của một hệ thống thường được NSD chọn làm tiêu chuẩn để phán xét hệ thống. Một giao diện thiết kế kém có thể làm cho NSD thực hiện hành động không đúng ý đồ thiết kế. Nếu các thông tin được

biểu diễn theo cách lẫn lộn và dễ hiểu nhầm thì NSD có thể nhận định sai ý nghĩa các mục chọn thông tin nên dễ sai lầm trong hành động.

Khi thiết kế giao diện người - máy phải tính đến yếu tố kinh nghiệm và khả năng ứng phó của NSD.

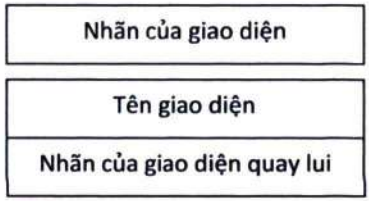
Trong thiết kế giao diện, giao diện của từng hệ con với các hệ con khác là được thiết kế và ghi thành tài liệu. Đặc tả giao diện không được mơ hồ và cho phép sử dụng hệ con đó mà không cần biết về phép toán trong hệ con.

Thiết kế giao diện là thể hiện sơ đồ phân rã quản lý chức năng chương trình bằng các form (khuôn dạng) màn hình. Thông qua các form đó, con người thực hiện trên máy các chức năng được mô tả trong sơ đồ luồng dữ liệu hệ thống.

4.4.2. Sơ đồ tương tác hệ thống giao diện

Trong thiết kế giao diện, đặc trưng xử lý đầu tiên của nó được đặc tả ở sơ đồ tương tác hệ thống giao diện.

Trong đặc tả tương tác giao diện, có 3 thành phần được biểu diễn trên một hình chữ nhật:



Hình 4.5. Đặc tả tương tác giao diện

Quy trình vẽ sơ đồ tương tác hệ thống giao diện:

Bước 1- Xác định các giao diện nhập liệu từ sơ đồ E_R;

Bước 2- Xác định các giao diện xử lý từ mô hình luồng dữ liệu hệ thống;

Bước 3- Tích hợp hai loại giao diện trên theo hướng giảm thiểu số lượng giao diện;

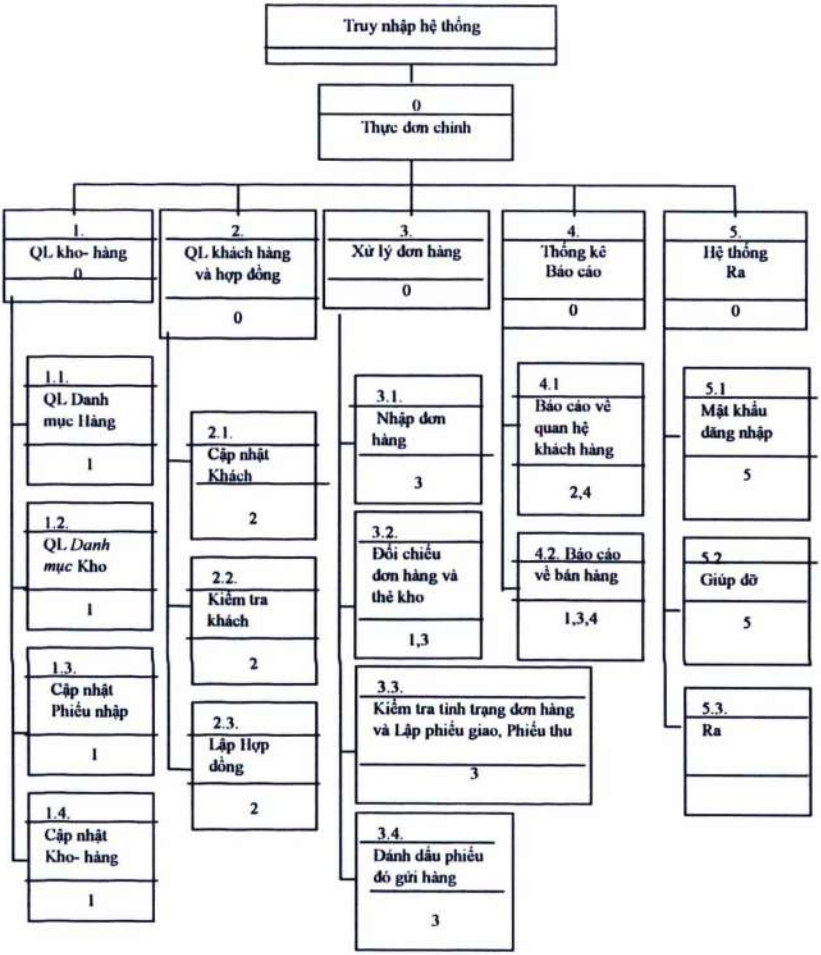
Bước 4- Xây dựng mô hình kiến trúc hệ thống bao gồm cấu trúc chương trình phần nghiệp vụ và thực đơn chính và con;

Bước 5- Đặc tả các giao diện;

Bước 6- Vẽ mẫu các giao diện và hướng dẫn sử dụng;

Bước 7- Vẽ sơ đồ tương tác hệ thống giao diện.

Ví dụ, sơ đồ tương tác hệ thống giao diện bài toán kinh doanh được đặc tả trong hình 4.6 [10].



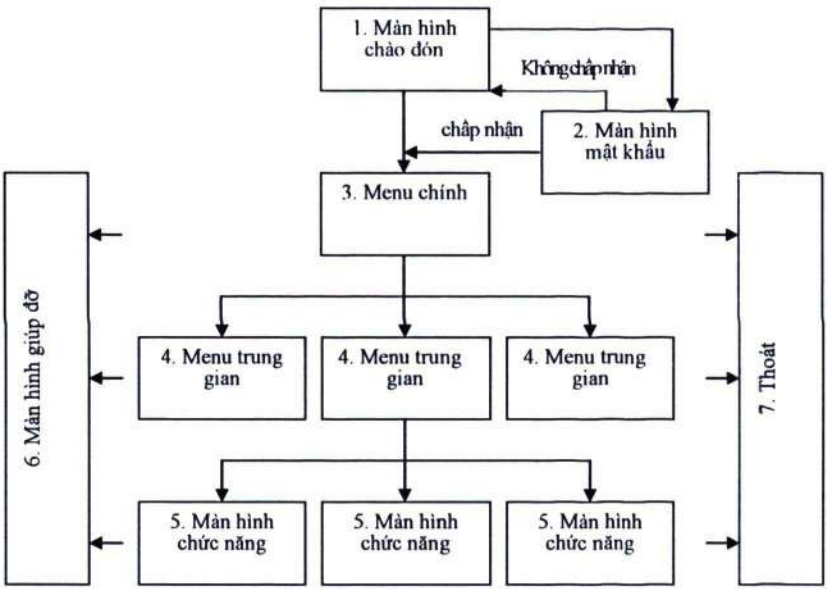
Hình 4.6. Sơ đồ tương tác hệ thống giao diện bài toán kinh doanh

Thiết kế giao diện có tầm quan trọng đặc biệt vì nó ảnh hưởng trực tiếp tới NSD. Trước hết, giao diện càng trực quan thì càng dễ sử dụng. Giao diện càng tốt thì việc đào tạo NSD càng thuận lợi và giảm được chi phí đào tạo. Lúc đó, NSD sẽ không cần giúp đỡ nhiều, thích sử dụng nó đồng thời có hứng thú giao tiếp với hệ thống.

Khi thiết kế giao diện, người phân tích hệ thống phải hoá thân trong vai người dùng, đồng thời phải làm nhiệm vụ tiếp thị để xác định kiểu "đóng gói" thích hợp cho HTTT. Người phân tích luôn phải nhớ rằng NSD có trình độ hiểu biết khác nhau về máy tính. Trong số họ, không ít người có nhiều kinh nghiệm, song cũng còn rất nhiều người không biết mấy về kỹ thuật CNTT.

4.4.3. Cấu trúc tương tác trên màn hình

Hệ thống tương tác bao gồm một lớp những giao diện, cho phép NSD lựa chọn một đường dẫn phù hợp dẫn tới những tác vụ đặc biệt muốn thực hiện.



Hình 4.7. Cấu trúc của hệ thống tương tác

Trong hình, các "vết" được đánh dấu (số 1,2,3...) tương ứng các lớp sau:

1- Giao diện chào đón: xuất hiện khi HTTT được kích hoạt ban đầu. Giao diện đặc trưng bao gồm: biểu tượng công ty, tên của hệ thống thông tin, và một lời chào thú vị nào đó.

2- Giao diện mật khẩu: Giao diện này ngăn ngừa những NSD không hợp pháp xâm nhập vào HTTT. Giao diện yêu cầu NSD nhập một mã nhận dạng để tiếp tục tiến trình.

3- Giao diện chính: Giao diện tương tác này cho phép NSD lựa chọn một vài ứng dụng trong HTTT.

4- Giao diện trung gian: Những giao diện này xúc tiến mô tả đường dẫn đến những chức năng thông tin yêu cầu.

5- Giao diện chức năng: Những giao diện tương tác cho phép NSD thực hiện chức năng như cập nhật hoặc xoá bản ghi.

6- Giao diện giúp đỡ: Những giao diện này hướng dẫn NSD làm sao để sử dụng một giao diện đặc biệt hoặc thực hiện 1 tác vụ đặc biệt.

7- Tuỳ chọn thoát khỏi: cho phép NSD thoát khỏi từ giao diện riêng hoặc đầu vào của HTTT.

4.4.4. Những nguyên tắc thiết kế

Shneiderman (1987) đã gợi ý tám "*quy tắc vàng*" cho thiết kế giao diện. Sau này người ta đã bổ sung thêm 1 thành 9 nguyên tắc như sau [10]:

1- Giữ cho đơn giản;

2- Giữ cho nhất quán;

3- Thiết kế các nhiệm vụ kiểu khép kín (sau khi chia nhiệm vụ thành các mô-đun nhỏ nhất, cung cấp thông tin phản hồi cho NSD để họ biết khi nào họ hoàn thành nhiệm vụ đó. Bằng cách này, NSD có thể sử dụng bộ nhớ tạm thời để thực hiện nhiệm vụ sau);

4- Hỗ trợ vùng điều khiển nội tại (làm cho NSD cảm giác họ chỉ điều khiển trong giao diện của họ, chứ không phải máy tính hoặc trình ứng dụng);

5- Cung cấp phím tắt cho NSD;

6- Xử lý lỗi một cách nhã nhặn;

7- Cho phép dễ lần ngược lại các hành động đã thực hiện. Không làm lúng túng người dùng. Thông báo lỗi cần phải là sáng sủa, ngắn, và đơn giản;

8- Sử dụng kỹ thuật gây chú ý thật hiệu quả. Điều ngạc nhiên khi nó xuất hiện hiếm khi xảy ra. Dùng các kỹ thuật để gây ngạc nhiên như độ chiếu sáng, tiếng chuông,...;

9- Đừng làm cho NSD lạc hướng. Người dùng cần phải luôn biết họ đang ở đâu và hướng tiếp tục đi.

4.4.5. Những yêu cầu chung đối với thiết kế giao diện

Với thiết kế giao diện có những yêu cầu đặc trưng sau đây [10]:

- Có nhiều dạng giao diện được sử dụng. Mỗi loại có những đặc tính và khả năng riêng của nó. Vì vậy việc thiết kế cần phù hợp với mỗi loại giao diện;

- Thiết kế phải phù hợp với chức năng cần thực hiện và với người sử dụng (thói quen, sở thích, trình độ);

- Chỉ tiêu cần đạt được:

+ *Dễ sử dụng*

+ *Dễ học (không phải nhớ nhiều)*

+ *Tốc độ thao tác nhanh*

+ *Sự thể hiện tinh vi*

+ *Kiểm soát tốt (hành vi của NSD)*

+ *Dễ phát triển;*

- Tuân thủ theo bảng phối màu của Courier [10]: bảng đã được sử dụng nhiều năm nay trong thiết kế biểu mẫu *(màu đen nền vàng, xanh lục nền trắng, đỏ nền trắng, xanh lơ nền trắng, trắng nền xanh lơ, đen nền trắng, vàng nền đen, trắng nền đỏ, trắng nền xanh lục, trắng nền đen, đỏ nền vàng, xanh lục nền đỏ, đỏ nền xanh lục);*

- Tuân thủ theo khung mẫu đặc tả giao diện trong thiết kế:

+ Tên giao diện

+ Nhãn của giao diện trong sơ đồ tương tác giao diện

+ Mục tiêu

+ Các chức năng đảm nhiệm

+ Ai sử dụng

+ Môi trường: (*Hệ điều hành, hệ quản trị CSDL,....)*

+ Mẫu thiết kế: (Hình kèm theo)

+ Các bảng dữ liệu sử dụng và các thuộc tính liên quan: danh sách các file dữ liệu;

+ Thao tác xử lý và các phương pháp xử lý được sử dụng;

+ Các ràng buộc toàn vẹn dữ liệu cần bảo đảm;

+ Hướng dẫn sử dụng;

+ Kết quả sau khi kết thúc giao diện;

+ Kết quả đánh giá thử nghiệm.

Một khi bản mẫu đầu tiên được xây dựng thì người thiết kế có thể thu thập rất nhiều dữ liệu định tính và định lượng sẽ trợ giúp cho việc đánh giá giao diện. Để thu thập dữ liệu định tính, có thể phân phát bảng hỏi cho người dùng bản mẫu.

Trong quá trình thiết kế, cần bảo đảm 3 phạm trù của hướng dẫn HCI, đã biết ở mức đại học, bao gồm [10]:

+ Tương tác chung;

+ Hiển thị thông tin;

+ Vào dữ liệu.

Trong quá trình thiết kế, cũng cần tuân theo 3 chiến thuật giao tác:

1- Cho phép NSD thoát ra một cách dễ dàng và nhanh chóng từ bất kỳ điểm nào của tiến trình giao tác;

2- Giúp NSD không bị lạc hướng. Bảo đảm rằng NSD luôn biết mình đang ở tiến trình nào vào thời điểm này và sắp đi đâu. Tâm lý chung của mọi người là sợ bị lạc hướng và bị mất phương hướng;

3- Thiết kế giao diện tương tác để dễ giao dịch. Mục tiêu nhằm cung cấp lượng thông tin nhanh hơn.

Phần lớn những người xây dựng ứng dụng phần mềm hiện đại đều cố cài đặt các giao diện trỏ và nhắp hướng cửa sổ. Việc tạo ra những giao diện như vậy không phải là việc dễ dàng. Vì vậy, cần chuẩn hoá giao diện sẽ làm lợi cho cả người phát triển và người dùng giao diện.

Mặc dầu một số chuẩn giao diện cạnh tranh đang được phát triển, chuẩn được dùng phổ biến nhất là X-Windows. Hệ thống X-Windows xác định một cú pháp và ngữ nghĩa cho thiết kế giao diện người-máy và cung cấp các công cụ để tạo ra hiển thị, cửa sổ và đồ hoạ cũng như giao thức để xử lý tài nguyên, tương tác thiết bị và giải quyết sự kiện. Một số biến thể và mở rộng của chuẩn hệ thống X-Windows đã được phát triển và dùng trên máy tính cá nhân và trạm làm việc dưới UNIX và các hệ điều hành khác.

4.4.6. Tiến trình thiết kế giao diện

Tiến trình thiết kế giao diện gồm 5 bước:

1- Bắt đầu với việc tạo ra các mô hình luồng dữ liệu hệ thống (dựa trên mô hình luồng dữ liệu nghiệp vụ)

2- Xác định các giao diện xử lý từ mô hình luồng dữ liệu hệ thống và đặc tả các tiến trình thực hiện chức năng hệ thống

3- Tích hợp với các giao diện cập nhật (nhờ mô hình khái niệm dữ liệu/sơ đồ E_R) để xây dựng mô hình kiến trúc hệ thống và sơ đồ tương tác hệ thống giao diện

4- Sử dụng các công cụ làm bản mẫu giao diện.

5- Cài đặt các giao diện thiết kế và đánh giá kết quả về chất lượng, tính tiện dụng.

Trong bước đầu tiên, việc xây dựng biểu đồ luồng dữ liệu hệ thống mang một ý nghĩa nhất định. Nó chỉ rõ tiến trình nào và kho dữ liệu nào máy làm, tiến trình nào, kho dữ liệu nào người làm.

Tiến trình thiết kế giao diện người dùng được xây dựng theo phương pháp "lặp", tức là một mô hình thiết kế được tạo ra, cài đặt như một bản mẫu, được người dùng xem xét và được thay đổi dựa trên lời góp ý của họ. Để thích nghi cách tiếp cận thiết kế lặp này, một lớp rộng các công cụ thiết kế giao diện và làm bản mẫu đã phát triển. Được gọi là các *bộ công cụ giao diện người dùng* hay *hệ thống phát triển giao diện người dùng* (UIDS), các công cụ này đưa ra các mô-đun hay đối tượng làm thuận tiện cho việc tạo ra cửa sổ, đơn, tương tác thiết bị, thông báo lỗi, chỉ lệnh và nhiều phần tử khác của môi trường tương tác.

4.5. Thiết kế mô-đun xử lý

Thiết kế thủ tục là biến đổi các thành phần cấu trúc thành đặc tả thủ tục phần mềm.

Chiến lược thiết kế thủ tục được đặt ra tuỳ thuộc vào từng cách tiếp cận. Ở đây, để đơn giản, chúng ta chỉ giới hạn trong cách tiếp cận hướng cấu trúc.

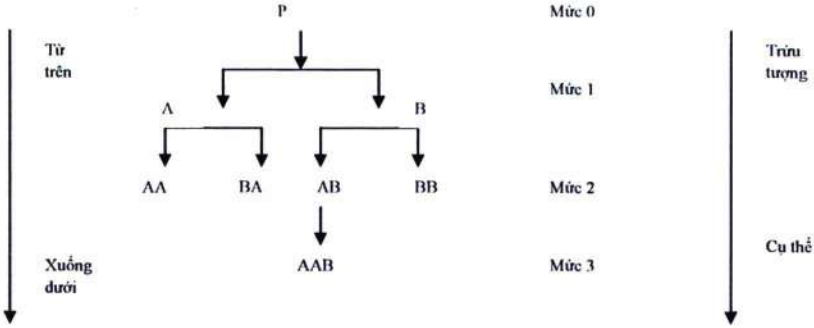
Trong hướng cấu trúc, chiến lược thiết kế thủ tục kiểu top-down *có một ý nghĩa rất quan trọng. Nó cho ta biết cách tổ chức giải quyết bài toán lớn, phức tạp như thế nào, cho biết cách thiết kế từ khái quát đến chi tiết, thể hiện bản chất của chiến lược thiết kế thủ tục.*

4.5.1. Cách tiếp cận top-down

Chiến lược thiết kế top-dow dựa trên cơ sở nguyên lý "*chia để trị*". Ý tưởng chính của phương pháp này là từ số liệu đầu vào của một vấn đề, một bài toán ban đầu, chúng ta chia thành các mô-đun nhỏ hơn một ít, chưa cần thiết kế chi tiết cho từng mô-đun.

Sau khi phân hoạch tổng thể xong ở mức 0, chúng ta lần lượt giải quyết từng mô-đun một. Với mỗi mô-đun này, đến lượt nó, ta lại có thể lại chia nhỏ thành các mô-đun nhỏ hơn (mức 1, 2, ...).

Việc giải các mô-đun nhỏ này sẽ góp phần vào việc giải quyết toàn bộ vấn đề. Như vậy chúng ta đi từ trừu tượng ban đầu để đến các cụ thể của mỗi mô-đun nhỏ.



Hình 4.8. Cách tiếp cận top-down

Cách thiết kế theo kiểu top-down giúp cho việc giải quyết bài toán được định hướng rõ ràng, tránh sa đà ngay vào các chi tiết phụ. Nó cũng là nền tảng cho việc lập trình có cấu trúc.

Thông thường đối với bài toán lớn, việc giải quyết nó phải do nhiều người cùng làm. Chính phương pháp mô-đun hoá sẽ cho phép bài toán tách thành các phần độc lập tạo điều kiện cho các nhóm giải quyết phần việc của mình mà không làm ảnh hưởng gì đến nhóm khác. Với chương trình được xây dựng trên cơ sở của các giải thuật được thiết kế theo cách này thì việc tìm hiểu cũng như sửa chữa chính lý sẽ dễ dàng hơn. Như vậy, tính ưu việt về tư duy ở đây thể hiện rõ nét tính hiệu quả của chiến lược thiết kế.

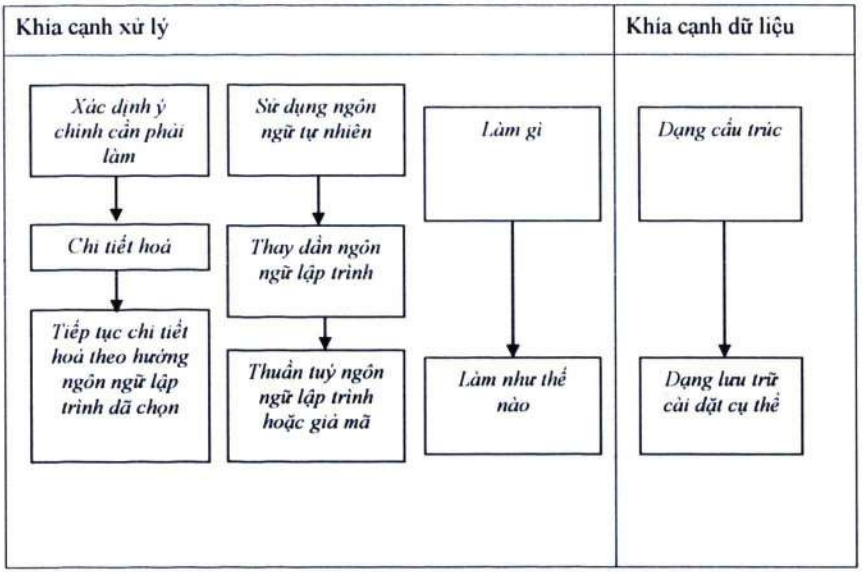
Việc phân bài toán thành những bài toán con như thế không phải là một việc làm dễ dàng. Chính vì vậy mà có các bài toán nhiệm vụ phân tích và thiết kế giải thuật bài toán đó còn mất nhiều thời gian và công sức hơn cả nhiệm vụ lập trình.

4.5.2. Phương pháp tinh chỉnh từng bước

Tinh chỉnh từng bước (Stepwise refinement) [7] là phương pháp thiết kế gắn liền với lập trình. Nó phản ánh tinh thần của quá trình mô hình hoá bài toán và thiết kế kiểu top-down.

Trong quá trình này dữ liệu cũng được "tinh chế" dần dần từ dạng *cấu trúc* đến *dạng lưu trữ cài đặt* cụ thể.

Nội dung các bước của phương pháp thiết kế thủ tục gắn liền với lập trình được thể hiện tóm tắt trong hình 4.9.



Hình 4.9. Các bước của phương pháp thiết kế thủ tục [10]

4.5.3. Đặc tả mô-đun

Trong cấu trúc phân cấp các thành phần của chương trình, mô-đun chương trình được hiểu như là một đơn vị khá nhỏ của hệ thống được định nghĩa bởi chức năng của nó. Nói cách khác, mô-đun là một thành phần tương đối độc lập trong hệ thống.

Một mô-đun chương trình có thể là một chương trình con dạng thủ tục, hàm, hoặc có thể là tập hợp một số câu lệnh trong chương trình

Nhiều ngôn ngữ lập trình có các UNIT, CLASS, OBJECT. Đó thực chất là các nhóm mô-đun chương trình (thường được gọi là các phương thức) tập hợp xung quanh một cấu trúc dữ liệu.

Một mô-đun thường có 4 tính chất cơ bản sau [10]:

1- Vào/ra:

+ Đầu vào: nhận được thông tin gì từ chương trình gọi nó,

+ Đầu ra: trả lại thông tin gì cho chương trình gọi nó;

2- Chức năng: chuyển đổi từ cái vào thành cái ra;

3- Cơ chế: phương thức cụ thể để biến cái vào thành cái ra;

4- Dữ liệu cục bộ: các ô nhớ hay cấu trúc dữ liệu dùng riêng.

Các thuộc tính vào/ra và chức năng tạo thành đặc trưng bề ngoài của một mô-đun. Còn cơ chế và dữ liệu cục bộ tạo thành các đặc trưng bên trong của mô-đun. Phương pháp thiết kế có cấu trúc lưu ý trước hết đến đặc trưng bề ngoài của mô-đun và lùi việc xem xét các đặc trưng bên trong của mô-đun lại sau.

Ngoài các thuộc tính cơ bản trên, mô-đun chương trình còn có một số thuộc tính khác cũng cần lưu ý khi thiết kế như tên của mô-đun (cho phép dùng tên để gọi mô-đun) và chỗ chiếm của nó trong văn bản chương trình hay bộ nhớ.

Ngoài các mô-đun được hình thành do việc chuyển đổi các chức năng xử lý, cần phải thêm một số mô-đun vào/ra và các mô-đun điều khiển.

Trong đặc tả mô-đun, các điều kiện cài đặt cụ thể như các tham số chuyển giao, các đối thoại, các xử lý tình huống, các thao tác, vào/ra, các truy nhập dữ liệu đều được mô tả đầy đủ.

Ví dụ 4.3

Đặc tả mô-đun "in phiếu đòi sách cho bạn đọc có sách quá hạn". Trong ví dụ này, thông tin về sách, bạn đọc và dữ liệu mượn trả được quan tâm [10].

1. Mô-đun: in phiếu đòi sách cho bạn đọc có sách quá hạn

- Dữ liệu vào:

  -số thẻ bạn đọc

  -ngày hiện thời (tự động)

  -tệp BANDOC, SÁCH, MUON TRA

- Dữ liệu ra:

  phiếu đòi và danh sách tài liệu mượn quá hạn cho từng bạn đọc

2. Mô tả xử lý:

Ng:=ngày hiện thời

Đối với mỗi bản ghi trong tệp BANDOC

a) K:=số thẻ

b) DSQH:= ∅

c) TT:=0

d) đối với mỗi bản ghi trong MUON TRA

IF (K=số thẻ) & (ngày trả=NULL) & (ngày hẹn trả<Ng)

THEN

TT:=TT+1;

DSQH(TT):= số hiệu sách

e) IF TT>0

THEN

- in tiêu đề phiếu đòi;
- Viết (số thẻ, tên bạn đọc);
- FOR I:=1 TO TT

Write (thông tin về SACH có Số hiệu sách=DSQH(I))

TÓM TẮT CHƯƠNG

Thiết kế phần mềm là một tiến trình chuyển hoá các yêu cầu thành một biểu diễn phần mềm.Thiết kế phần mềm nằm ở trung tâm kỹ thuật của tiến trình kỹ nghệ phần mềm và được áp dụng bất kể tới khuôn cảnh phát triển được sử dụng.

Thành quả của mỗi hoạt động thiết kế là một đặc tả. Đặc tả này có thể là một đặc tả trừu tượng, hình thức và được tạo ra để làm rõ các yêu cầu, nó cũng có thể là một đặc tả về một phần nào đó của hệ thống phải được thực hiện như thế nào.

Theo quan điểm quản lý dự án, thiết kế phần mềm được tiến hành theo hai bước: 1-*Thiết kế sơ bộ; 2-Thiết kế chi tiết.*

Trong phạm vi thiết kế sơ bộ và chi tiết, có xuất hiện một số hoạt động thiết kế khác nhau. Bên cạnh việc thiết kế dữ liệu, kiến trúc và thủ tục, nhiều ứng dụng hiện đại có hoạt động *thiết kế giao diện* phân biệt. Thiết kế giao diện lập ra cách bố trí và cơ chế tương tác cho tương tác người máy.

Thiết kế dữ liệu là việc chuyển mô hình lĩnh vực thông tin đã được tạo ra trong bước phân tích thành các cấu trúc dữ liệu sẽ cần cho việc cài đặt phần mềm. Thiết kế dữ liệu bao gồm thiết kế dữ liệu mức logic và thiết kế vật lý.

Thiết kế kiến trúc là định nghĩa ra mối quan hệ giữa các thành phần cấu trúc chính của chương trình. Trong thiết kế kiến trúc hệ thống, các hệ con tạo nên hệ tổng thể và các quan hệ của chúng được minh định và ghi thành tài liệu. Trong thiết kế kiến trúc hệ thống, người ta tập trung vào thiết kế cấu trúc chương trình.

Thiết kế giao diện là thể hiện ý tưởng chuyển các chức năng được mô tả bằng sơ đồ luồng hệ thống sang dạng một màn hình thao tác để qua đó con người tác động nhằm thực hiện các chức năng đó trên máy. Tính dễ sử dụng giao diện của một hệ thống thường được NSD chọn làm tiêu chuẩn để phán xét hệ thống.

Trong thiết kế giao diện, đặc trưng xử lý đầu tiên của nó được đặc tả ở sơ đồ tương tác hệ thống giao diện.

Shneiderman (1987) đã gợi ý tám "*quy tắc vàng*" cho thiết kế giao diện, trong đó nguyên tắc số 1 là "giữ cho đơn giản". Trong thiết kế giao diện, phải hết sức chú ý đến bảng phối màu của Courier, 3 phạm trù của hướng dẫn HCI và 3 chiến thuật giao tác.

Tiến trình thiết kế giao diện gồm nhiều bước, bắt đầu với việc tạo ra các mô hình luồng dữ liệu hệ thống (dựa trên mô hình luồng dữ liệu nghiệp vụ).

Thiết kế thủ tục là biến đổi các thành phần cấu trúc thành đặc tả thủ tục phần mềm. Chiến lược thiết kế thủ tục được đặt ra tuỳ thuộc vào từng cách tiếp cận. Trong hướng cấu trúc, chiến lược thiết kế thủ tục kiểu top-down có một ý nghĩa rất quan trọng. Nó cho ta biết cách tổ chức giải quyết bài toán lớn, phức

tạp như thế nào, cho biết cách thiết kế từ khái quát đến chi tiết, thể hiện bản chất của chiến lược thiết kế thủ tục. Tinh chỉnh từng bước là phương pháp thiết kế gắn liền với lập trình. Nó phản ánh tinh thần của quá trình mô hình hoá bài toán và thiết kế kiểu top-down.

Mô-đun là một thành phần tương đối độc lập trong hệ thống. Một mô-đun chương trình có thể là một chương trình con dạng thủ tục, hàm, hoặc có thể là tập hợp một số câu lệnh trong chương trình.

Tùy bài toán cụ thể mà có thể ánh xạ mỗi chức năng xử lý biến thành một mô-đun, có thể tổ hợp một số chức năng xử lý thành một mô-đun, hoặc ngược lại, phân rã một chức năng xử lý thành một số mô-đun. Ngoài các mô-đun được hình thành do việc chuyển đổi các chức năng xử lý, cần phải thêm một số mô-đun vào/ra và các mô-đun điều khiển.

CÂU HỎI CHƯƠNG 4

4.1. Tại sao nói Thiết kế phần mềm nằm ở trung tâm kỹ thuật của tiến trình kỹ nghệ phần mềm?

4.2. Theo quan điểm quản lý dự án, thiết kế phần mềm được tiến hành theo mấy bước?

4.3.Tại sao thiết kế CSDL cần phân ra 2 mức là logic và vật lý?

4.4. Đối với các quy tắc phân rã thành hệ lược đồ quan hệ đạt chuẩn 1NF, 2NF, 3NF tương ứng, nếu có m nhóm thuộc tính lặp tách biệt hay m phụ thuộc bộ phận hay m thuộc tính cầu dựa theo tinh thần các quy tắc trên thì chúng ta có thể tách thành bao nhiêu lược đồ?

4.5. Trong thiết kế kiến trúc hệ thống, người ta tập trung vào thiết kế cái gì?

4.6. Người ta thường xây dựng sơ đồ cấu trúc chương trình từ sơ đồ luồng dữ liệu hệ thống, sơ đồ này khác sơ đồ luồng dữ liệu nghiệp vụ ở điểm nào?

4.7.Tại sao chúng ta cần phải vẽ sơ đồ tương tác hệ thống giao diện?

4.8. Tại sao thiết kế giao diện lại có tầm quan trọng đặc biệt đối với NSD?

4.9. Phương pháp "lặp" trong tiến trình thiết kế giao diện người dùng nghĩa là thế nào?

4.10. Trong hướng cấu trúc, chiến lược thiết kế thủ tục *kiểu top-down* có một ý nghĩa quan trọng như thế nào?

BÀI TẬP CHƯƠNG 4

Bài tập 4.1

Đặc tả phân tích yêu cầu dữ liệu mức khái niệm và đặc tả thiết kế dữ liệu mức logic dựa trên cơ sở 2 chứng từ sau?:

1. ĐONMUAHANG

Số : Ngày:

Tên Nhà cung cấp

Địa chỉ Nhà cung cấp:

STT	Tên hàng	Đơn vị tính	Đơn giá	Số lượng	Thành tiền

3. PHIEUNHAPKHO

Số: Ngày:

Mã NCC

Địa chỉ NCC:

STT	Tên hàng	Mã hàng	Đơn vị tính	Số lượng

Bài tập 4.2

Đặc tả thiết kế thủ tục lập một chương trình sắp xếp một dãy n số nguyên khác nhau theo thứ tự tăng dần theo phương pháp tinh chỉnh từng bước?

Bài tập 4.3

Đặc tả tiến trình « Kiểm tra khách » trong bài tập 1.1 chương 1?

Bài tập 4.4

Đặc tả mô-đun giải phương trình bậc nhất ax+b=0?

Chương 5
ĐẶC TẢ THIẾT KẾ KIẾN TRÚC

Kiến trúc không phải thành phần hoạt động, nhưng có tác động sâu rộng đến toàn bộ tiến trình phát triển phần mềm sau này. Nó là một mô tả về phần mềm cho phép các kỹ sư phần mềm hình dung được hệ thống cần xây dựng một cách đầy đủ và chính xác, lựa chọn được một kiến trúc hiệu quả nhất trong điều kiện có thể, giảm thiểu được chi phí hay rủi ro có thể do kiến trúc gây ra.

Chương này tập trung trình bày một nội dung cốt lõi của tiến trình phát triển phần mềm là thiết kế kiến trúc phần mềm.

Học viên cần nắm vững khái niệm kiến trúc phần mềm, hiểu rõ các mô hình kiến trúc ứng dụng, một số phương pháp tạo kiến trúc tiêu biểu, tiến trình thiết kế kiến trúc hệ thống phần mềm, một số tiêu chuẩn thiết kế kiến trúc.

Học viên cần nâng cao kỹ năng phân tích đặc tả yêu cầu phần mềm thành đặc tả kiến trúc phần mềm và kỹ năng thực hành thiết kế kiến trúc nhằm bảo đảm chất lượng thiết kế.

5.1. Kiến trúc phần mềm và đặc tả thiết kế kiến trúc

5.1.1. Định nghĩa kiến trúc phần mềm

Shaw và Garlan [21] đã đưa ra khái niệm về kiến trúc phần mềm: "*Kiến trúc phần mềm ngầm chỉ một cấu trúc tổng thể của phần mềm và qua đó cung cấp một sự tích hợp về mặt khái niệm của một hệ thống*".

Kiến trúc hệ thống *là khái niệm "mức cao nhất" về hệ thống trong môi trường của nó. Ở đây,* "mức cao nhất" thể hiện tính trừu tượng hóa từ các chi tiết về thiết kế, triển khai để tập trung vào tính tổng thể của hệ thống (IEEE).

Theo Bass, Clements và Kazman [23]: "*Kiến trúc phần mềm của một chương trình hoặc hệ thống máy tính là cấu trúc hoặc các cấu trúc của hệ*

thống, bao gồm các thành phần phần mềm cùng các thuộc tính nhận biết được của chúng và các mối quan hệ giữa các thành phần đó".

Theo Sommerville [25]: *"Kiến trúc phần mềm (software architecture) được hiểu như một cấu trúc tổng thể của phần mềm và qua đó cung cấp một sự tích hợp về mặt khái niệm của một hệ thống".*

Theo nghĩa rộng, *kiến trúc phần mềm biểu diễn các thành phần lớn, cốt lõi của hệ thống và mối quan hệ giữa chúng với nhau được nhìn theo những quan điểm khác nhau* (cấu trúc, hành vi, dữ liệu,...). Ở dạng đơn giản nhất, kiến trúc phần mềm là cấu trúc phân cấp của các thành phần chương trình qua đó thể hiện sự tương tác giữa chúng với nhau và các cấu trúc dữ liệu mà chúng sử dụng [16].

Kiến trúc không phải thành phần hoạt động, nhưng có tác động sâu rộng đến toàn bộ tiến trình phát triển phần mềm sau này. Nó là một mô tả về phần mềm cho phép các kỹ sư phần mềm hình dung được hệ thống cần xây dựng một cách đầy đủ và chính xác, lựa chọn được một kiến trúc hiệu quả nhất trong điều kiện có thể, giảm thiểu được chi phí hay rủi ro có thể do kiến trúc gây ra [9].

Kiến trúc phần mềm được thể hiện qua hai đặc trưng quan trọng của chương trình máy tính:

1- Cấu trúc cấp bậc điều khiển của các thành phần thủ tục (mô-đun) (hoặc cấu trúc chương trình);

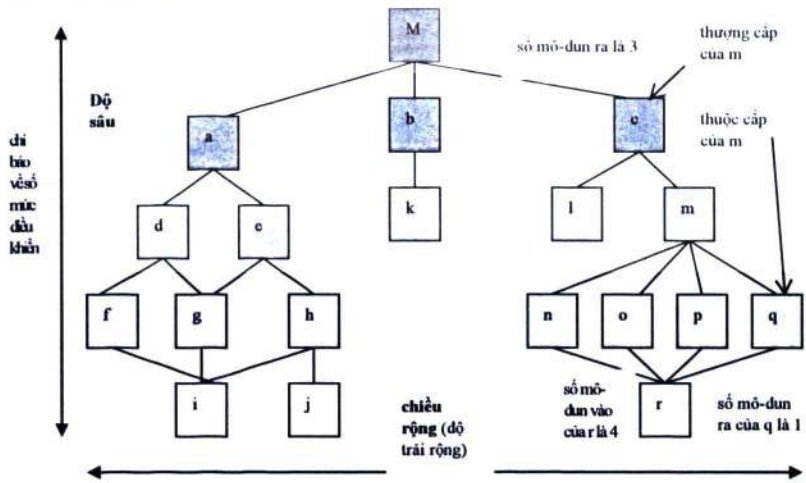
2- Cấu trúc dữ liệu.

Một vấn đề có thể được thoả mãn bởi nhiều cấu trúc khác nhau. Phương pháp thiết kế phần mềm có thể được dùng để suy ra cấu trúc, nhưng vì từng cấu trúc lại dựa trên các khái niệm nền tảng khác nhau về thiết kế "tốt", cho nên từng phương pháp sẽ phát sinh trong một cấu trúc khác biệt đối với cùng tập các yêu cầu phần mềm. Không có câu trả lời dễ dàng đối với câu hỏi "Cái nào tốt nhất?". Tuy nhiên, có những đặc trưng về cấu trúc mà ta có thể kiểm tra để xác định chất lượng tổng thể [9].

Cấp bậc điều khiển, còn gọi là *cấu trúc chương trình*, biểu thị cho cách tổ chức của các thành phần chương trình (mô-đun). Người ta dùng các ký pháp

khác nhau để biểu diễn cho cấp bậc điều khiển. Thông dụng nhất là biểu đồ kiểu cây. Trong đó *độ sâu* và *chiều rộng* đưa ra một chỉ báo về số mức điều khiển và độ trải rộng toàn bộ của điều khiển tương ứng. *Số mô-đun ra* là một độ đo để đo số các mô-đun trực tiếp bị điều khiển bởi các mô-đun khác. *Số mô-đun vào* chỉ ra cách thức mô-đun trực tiếp điều khiển một mô-đun đã cho.

Đối với sơ đồ phức tạp, khó biểu diễn dưới dạng sơ đồ phân cấp, người ta thường dùng đường thẳng có mũi tên để thể hiện sự điều khiển, mô-đun điều khiển gắn với gốc đường thẳng có mũi tên. Lúc đó, người ta gọi nó là lược đồ cấu trúc chương trình.



Hình 5.1. Thuật ngữ cấu trúc [9]

Cấu trúc dữ liệu: biểu diễn mối quan hệ logic giữa các phần tử dữ liệu riêng lẻ. Vì cấu trúc thông tin sẽ luôn luôn ảnh hưởng tới thiết kế thủ tục cuối cùng nên cấu trúc dữ liệu cũng quan trọng như cấu trúc chương trình để biểu thị kiến trúc phần mềm.

Cấu trúc dữ liệu mang một ý nghĩa lớn, nó khống chế cách tổ chức, các phương pháp thâm nhập, mức độ kết hợp và các phương án xử lý thông tin.

Điều quan trọng cần lưu ý rằng cấu trúc dữ liệu, giống như cấu trúc chương trình, có thể được biểu diễn ở các mức trừu tượng khác nhau.

5.1.2. Vai trò và tầm quan trọng của kiến trúc phần mềm

Kiến trúc có một vai trò quan trọng trong phát triển phần mềm. Bass [19] và đồng nghiệp của ông đã chỉ rõ ba vai trò của kiến trúc phần mềm:

1- Là công cụ giao tiếp giữa những người có liên quan đến phát triển hệ thống.

2- Để phân tích hệ thống: *Khi đưa ra kiến trúc hệ thống ở giai đoạn đầu, nhà phân tích có thể tiến hành những quyết định thiết kế có tác động sâu sắc đến hệ thống sau này như tính hoàn thiện, độ ổn định, tính bảo trì được,...*

3- Giúp sử dụng lại ở quy mô lớn: *Kiến trúc thu được có thể dùng cho các hệ thống có yêu cầu tương tự và vì vậy có thể dùng để hỗ trợ việc sử dụng lại với quy mô lớn.*

5.1.3. Đặc tả thiết kế kiến trúc

Mục tiêu chủ yếu của thiết kế kiến trúc là phát triển một cấu trúc chương trình theo mô-đun và biểu diễn mối quan hệ điều khiển giữa các mô-đun. Bên cạnh đó, thiết kế kiến trúc còn bao gồm cấu trúc chương trình và cấu trúc dữ liệu, xác định các giao diện làm cho dữ liệu chảy qua toàn bộ chương trình. Mô-đun là một thành phần hệ thống. Nó cung cấp một dịch vụ hoặc phục vụ cho các thành phần khác và không hoạt động như một hệ thống [23].

Thiết kế kiến trúc là giai đoạn đầu của quá trình thiết kế. Nó biểu diễn sự kết nối giữa đặc tả yêu cầu và các tiến trình thiết kế. Nó thường tiến hành song song với các hoạt động đặc tả phần mềm và bao gồm việc xác định các thành phần hệ thống và giao tiếp giữa chúng.

Có nhiều phương pháp thiết kế kiến trúc. Mặc dầu mỗi phương pháp đều có cách tiếp cận khác nhau để suy ra kiến trúc, nhưng tất cả đều thừa nhận tầm quan trọng của cái nhìn chung về phần mềm.

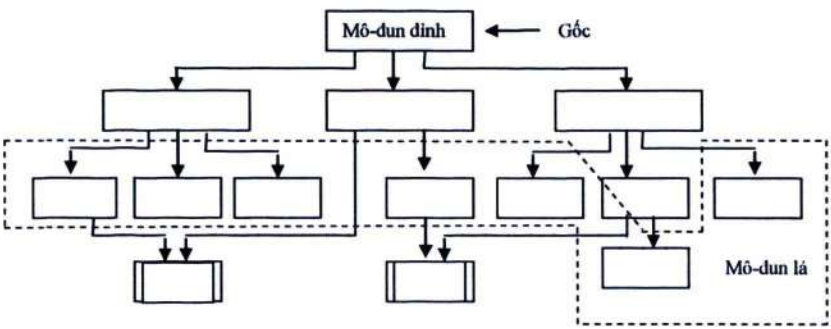
Thiết kế kiến trúc tập trung vào việc biểu diễn cấu trúc của các thành phần phần mềm, các thuộc tính của chúng và các tương tác giữa chúng [23].

Thiết kế kiến trúc hệ thống bao gồm việc thiết kế các hệ con tạo nên hệ tổng thể, thiết kế quan hệ của các hệ con này. Các hệ con và các mối quan hệ

đó được minh định và được ghi thành tài liệu. Một hệ thống lớn luôn có thể phân rã thành các hệ thống con độc lập tương đối với nhau. *Tiến trình xác định các hệ thống con của một hệ thống và thiết lập một khung làm việc cho việc điều khiển và giao tiếp giữa chúng gọi là thiết kế kiến trúc.* Kết quả của hoạt động này là một tài liệu thiết kế kiến trúc [9].

Trong thiết kế kiến trúc, một thành phần phần mềm có thể là một mô-đun chương trình, nhưng chúng có thể mở rộng thành những CSDL và phần đệm cho phép cấu hình một mạng Client-Server. Các thuộc tính của thành phần là những đặc trưng cần thiết để hiểu cách mà các thành phần tương tác với nhau như thế nào. Còn các thuộc tính bên trong ở mức kiến trúc (nghĩa là chi tiết của một thuật toán) chưa được mô tả. Mối quan hệ giữa các thành phần có thể đơn giản như thủ tục gọi từ một mô-đun đến một mô-đun khác hoặc một tổ hợp giao thức truy nhập vào một CSDL.

Biểu đồ chung nhất để biểu diễn mô hình kiến trúc của một hệ thống là biểu đồ phân cấp. Biểu đồ mô hình kiến trúc là một biểu diễn bằng đồ thị các thành phần (gọi chung là các mô-đun) của hệ thống trong mối liên hệ với nhau xét về mặt vận động của dữ liệu cũng như sự liên kết giữa các bộ phận trong mỗi chương trình. Biểu diễn đồ thị của mô hình kiến trúc là một biểu đồ phân cấp, giống như một cây hay một biểu đồ tổ chức có đỉnh và nhiều mức, gồm các mô-đun lồng nhau. Quá trình xây dựng biểu đồ của mô hình kiến trúc một hệ thống là quá trình đưa các biểu đồ luồng dữ liệu vào một trong các dạng cấu trúc thành phần hệ thống theo một số nguyên tắc của thiết kế chương trình.



Hình 5.2. Mô hình kiến trúc hệ thống chương trình [17]

Một mô-đun trong biểu đồ là một đơn vị độc lập tương đối của một hệ thống được xác định bằng chính chức năng của nó. Các mô-đun cũng có thể chứa các thành phần của nó và đến lượt mình mỗi thành phần cũng có thể là một mô-đun. Các mô-đun được thực hiện như một đơn vị độc lập có điểm vào và điểm ra của nó. Mọi chương trình máy tính thường được xây dựng từ một số mô-đun.

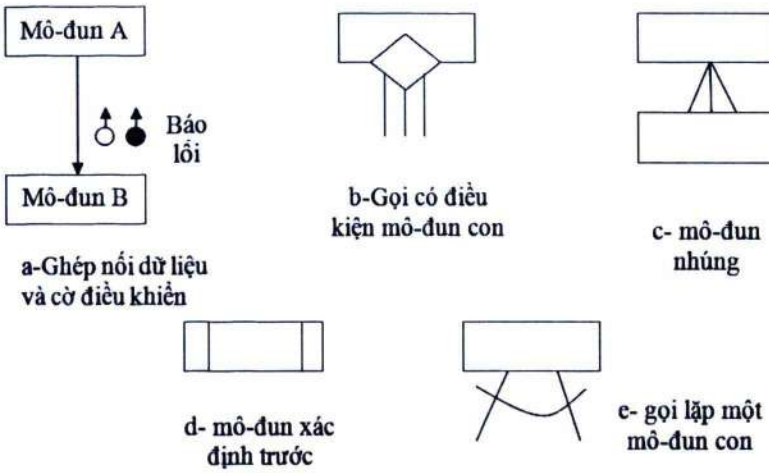
Trong biểu đồ cấu trúc, ở gốc là một mô-đun liên kết. Nó có nhiệm vụ phối hợp các mô-đun ở các mức sau bằng "thao tác gọi" đến các mô-đun đó. Đối với một hệ thống đang hoạt động, có thể hình dung mô-đun gốc là thực đơn chính và mỗi mô-đun ở mức dưới là một thực đơn con, gồm một số tùy chọn. Mỗi tùy chọn có thể là một mô-đun. Một mô-đun có thể gọi đến một mô-đun mức thấp hơn khi cần thực hiện một chức năng của nó.

Các mô-đun ở mức thấp nhất trong biểu đồ hình cây gọi là mô-đun lá. Nó thực hiện một nhiệm vụ cụ thể và không gọi đến mô-đun nào khác. Các mô-đun mức giữa có nhiệm vụ phối hợp và kiểm soát sự hoạt động của các mô-đun mức thấp hơn và được gọi là mô-đun liên kết.

Các mô-đun trong một biểu đồ cấu trúc giao tiếp với mỗi mô-đun khác thông qua tham số. Các tham số là dữ liệu được biểu diễn như một "ghép nối dữ liệu" hay các "cờ điều khiển".

Một ghép nối dữ liệu cho biết dữ liệu đã được trao đổi giữa hai mô-đun. Nó được biểu diễn bằng một hình tròn nhỏ có gắn mũi tên đi ra. Hướng mũi tên chỉ hướng di chuyển của ghép nối dữ liệu. Một cờ điều khiển để chỉ dữ liệu điều khiển, biểu diễn giống như ghép nối dữ liệu nhưng với hình tròn đen. Nó cũng di chuyển giữa hai mô-đun.

Cờ điều khiển biểu diễn thông tin mà hệ thống cần cho hoạt động xử lý, nhưng nó không bao giờ bị xử lý. Ví dụ, một cờ điều khiển mang thông báo như "cuối tệp" hay "giá trị ra ngoài giới hạn". Một cờ điều khiển không biểu diễn thông tin mà là một mô-đun "ra lệnh" cho một mô-đun khác làm gì.



Hình 5.3. Một số ký pháp có thể sử dụng trong mô hình kiến trúc

Biểu đồ cấu trúc được đọc từ trái sang phải và từ trên xuống dưới, thứ tự gọi các mô-đun được xác định bằng các ký hiệu thông tin di chuyển (ghép nối dữ liệu, cờ điều khiển) đặt vào các mô-đun và theo hướng chỉ ra của nó. Xét về mặt cấu trúc, các mô-đun trong biểu đồ cấu trúc thường phân làm hai loại là mô đun liên kết (liên kết các mô-đun con dưới nó) và mô-đun xử lý (tiến hành các xử lý thực sự). Các mô-đun liên kết tương ứng với các bảng thực đơn, còn các mô-đun xử lý chính là nội dung của một thực đơn hay là thành phần được gọi trong một mô-đun nào đó [17].

5.2. Các mô hình kiến trúc ứng dụng

Kiến trúc phần mềm được mô tả ở dạng các mô hình khác nhau. Các mô hình kiến trúc được phát triển thường có thể nhìn nhận dưới một số mặt [25] sau:

1- *Mô hình kiến trúc tĩnh* chỉ ra các hệ con hay các thành phần được phát triển như một đơn vị độc lập.

2- Mô hình tiến trình động chỉ ra hệ thống được tổ chức thành các tiến trình vận hành.

3- Mô hình giao diện xác định các giao diện đưa ra các dịch vụ mà mỗi hệ con cung cấp thông qua giao diện của chúng.

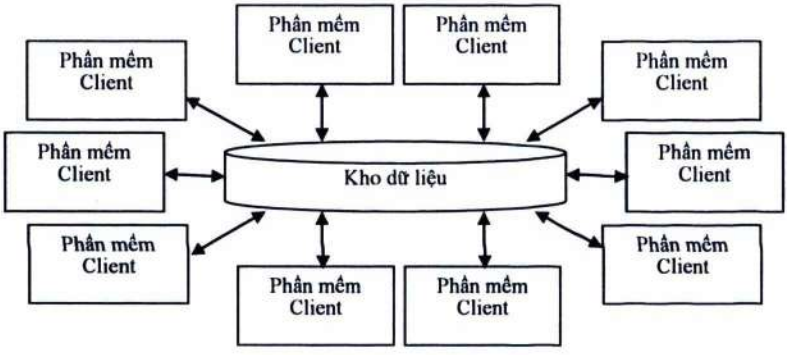
4- *Mô hình liên kết* chỉ ra mối liên kết giữa các hệ con hay giữa các thành phần.

5.2.1. Kiến trúc phần mềm theo hướng dữ liệu tập trung hay phân tán

Trong một hệ thống, các hệ thống con cần trao đổi dữ liệu với nhau để làm việc hiệu quả. Có hai cách cơ bản để tổ chức thực hiện được điều này: tổ chức dữ liệu tập trung tại một chỗ và tổ chức dữ liệu phân tán ở nhiều vị trí.

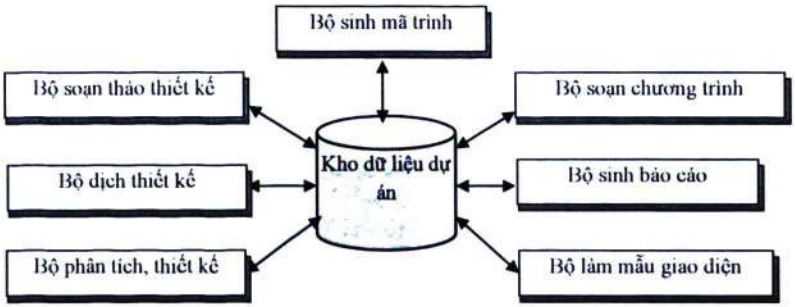
5.2.1.1. Mô hình kiến trúc kho dữ liệu (repository model)

Mô hình kho dữ liệu là một cách tổ chức dữ liệu tập trung (Data-Centered Architecture). Tất cả dữ liệu chia sẻ được lưu giữ tại một CSDL trung tâm, sao cho tất cả các hệ con đều có thể truy nhập.



Hình 5.4. Cách tổ chức dữ liệu tập trung

Mô hình này thích hợp với các ứng dụng khi dữ liệu do mỗi hệ con sinh ra được tập trung lại để các hệ con khác có thể sử dụng, chẳng hạn các hệ CAM/CAD hay các bộ công cụ CASE. Hình 5.5 là một ví dụ kiến trúc kho dữ liệu tập trung.



Hình 5.5. Kiến trúc trong bộ CASE tích hợp

Kiến trúc này có ưu điểm:

1) Tiện lợi cho chia sẻ dữ liệu lớn;

2) Các phân hệ không cần quan tâm tổ chức dữ liệu;

nhưng cũng có nhược điểm:

3) Các phân hệ phải thống nhất mô hình dữ liệu;

4) Khó thay đổi cấu trúc dữ liệu;

5) Các phân hệ không thể đưa ra chính sách riêng;

6) Khó khăn trong quản lý giao dịch.

5.2.1.2. Mô hình kiến trúc khách - dịch vụ (Client-Server)

Mô hình kiến trúc khách - dịch vụ (*Client-Server Architecture)* là một mô hình hệ thống phân tán, ở đó dữ liệu và xử lý được phân bố trên các máy khác nhau. Thành phần chính của mô hình này bao gồm:

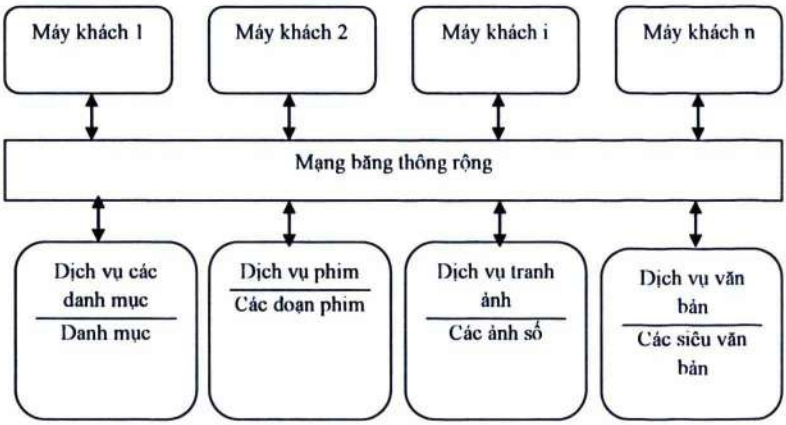
- Các máy dịch vụ *cung cấp dịch vụ cho các hệ con khác.*

- *Các máy khách* gọi để sử dụng dịch vụ từ các máy dịch vụ

- *Mạng liên kết* cho phép các máy khách truy nhập đến máy dịch vụ.

Các máy khách cần biết tên của các máy dịch vụ đang sẵn sàng và tên các dịch vụ mà nó cung cấp. Ngược lại, máy dịch vụ không cần biết định danh của máy khách hoặc có bao nhiêu máy khách. Các máy khách truy cập sử dụng dịch vụ mà máy dịch vụ cung cấp thông qua thủ tục gọi từ xa. Một lợi thế của mô hình này là mô hình phân tán, có thể dễ dàng thêm vào nó một máy dịch vụ

và tích hợp nó với các máy còn lại, hoặc nâng cấp máy dịch vụ mà không gây ảnh hưởng đến các phần khác của hệ thống.



Hình 5.6. Kiến trúc hệ thống thư viện phim và tranh ảnh

Đặc điểm của kiến trúc Client-Server là:

- Hệ thống được phân tán về dữ liệu và xử lý thông qua quan hệ các thành phần;

- Các máy dịch vụ (server) độc lập cung cấp các dịch vụ đặc biệt như: in ấn, quản lý dữ liệu;

- Các máy khách (Client) sử dụng dịch vụ Server;

- Hệ thống mạng để Client truy cập dịch vụ Server.

Đây là mô hình phát triển ứng dụng phổ biến nhưng bên cạnh những ưu điểm cũng còn những nhược điểm như:

- Khó tích hợp dữ liệu;

- Cần cơ chế bảo toàn dữ liệu cho từng Server.

Chúng ta sẽ nghiên cứu sâu hơn mô hình kiến trúc này trong chương sau.

5.2.1.3. Mô hình kiến trúc phân tầng

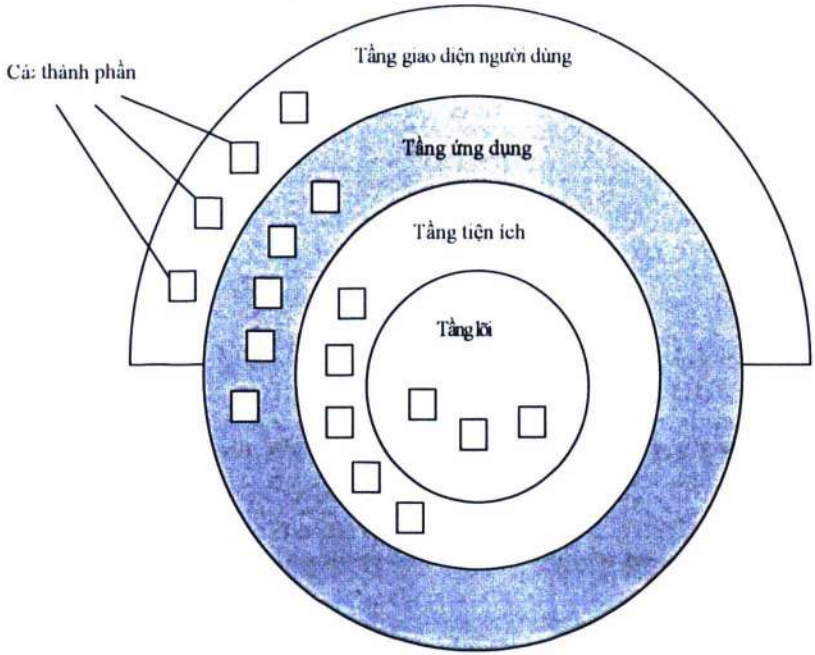
Mô hình kiến trúc phân tầng (Layered Architecture / Abstract machine model) là:

- Mô hình hóa tương tác giữa các hệ con;

- Thể hiện sự phân rã hệ thống thành các tầng, mỗi tầng cung cấp một tập các dịch vụ;

- Mỗi tầng thay đổi thì chỉ ảnh hưởng tầng liền kề, do vậy, có sự phát triển tăng trưởng;

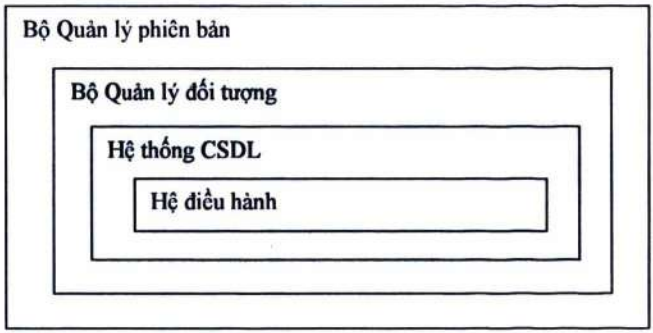
- Mô hình khó xây dựng.



Hình 5.7. Mô hình kiến trúc phân tầng

Ví dụ 5.1:

Trong hệ thống quản lý phiên bản, người ta xây dựng kiến trúc của nó như hình 5.8.



Hình 5.8. Kiến trúc hệ thống quản lý văn bản

5.2.2. Kiến trúc phần mềm theo hướng điều khiển tập trung

Mô hình điều khiển bao gồm:

1- *Kiến trúc gọi và trả lại* (*Call and Return Architecture*) và quản lý tập trung (*manager centralised Architecture)*;

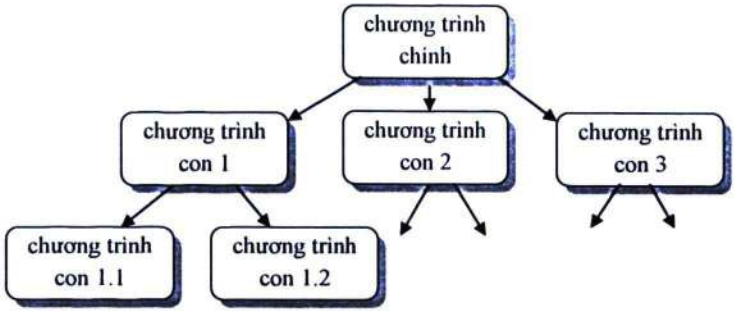
2- *Kiến trúc xử lý hướng sự kiện* (*event-based Architecture*)

Để làm việc như một hệ thống, các hệ con cần được điều khiển sao cho dịch vụ được gửi đến đúng chỗ, đúng thời gian. Các mô hình điều khiển ở mức kiến trúc liên quan đến luồng điều khiển giữa các hệ con.

Phương thức điều khiển đầu tiên được quan tâm là *điều khiển tập trung (centralised model). Một* hệ con có trách nhiệm điều khiển, khởi động hay dừng các hệ con khác. Nó cũng có thể chuyển một điều khiển đến một hệ con khác và chờ để trách nhiệm điều khiển quay trả lại nó.

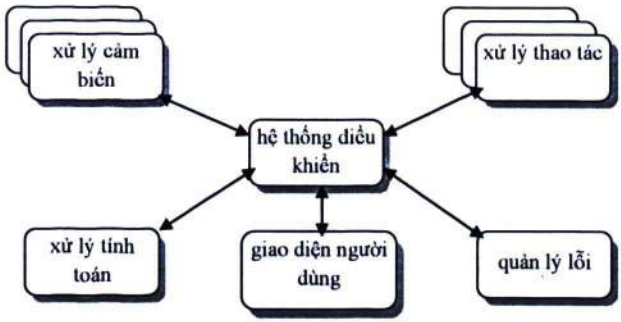
Trong mô hình điều khiển tập trung, một hệ con được thiết kế như bộ điều khiển toàn hệ thống. Các mô hình điều khiển tập trung có thể thuộc vào hai lớp:

- Mô hình gọi-trả lại (Call and Return model): *Nó sử dụng một hệ con để điều khiển, khởi động và dừng các hệ thống con khác, truyền điều khiển theo chiều từ trên xuống dưới. Mô hình này chỉ dùng cho các hệ thống tuần tự.*



Hình 5.9. Minh họa mô hình điều khiển gọi - trả lại

Mô hình quản lý (manager model) được áp dụng cho các hệ tương tranh. Một hệ thống thành phần được thiết kế như một hệ quản lý và điều khiển sự bắt đầu, dừng hay phối hợp các tiến trình của các hệ thống khác.



Hình 5.10. Mô hình quản lý tập trung cho hệ thời gian thực

Hình 5.10. minh họa một mô hình điều khiển tập trung một hệ tương tranh. Mô hình này thường được sử dụng hệ thống thời gian thực "mềm", không có ràng buộc quá chặt về thời gian.

5.2.3. Kiến trúc phần mềm theo hướng sự kiện

Mô hình điều khiển dựa trên sự kiện (event-based model) *cũng là một mô hình điều khiển ở mức kiến trúc liên quan đến luồng điều khiển giữa các hệ con. Thông tin điều khiển được chuyển đến một số hệ con, mỗi hệ con có thể đáp ứng sự kiện sinh ra từ bên ngoài. Các sự kiện có thể đến từ hệ con khác hay từ môi trường của hệ thống.*

Kiến trúc xử lý hướng sự kiện là cơ chế gửi yêu cầu xử lý đến các hệ con. Nó bao gồm hai mô hình chính:

1- Điều khiển quảng bá (Broadcast);

2- Điều khiển hướng ngắt (Interrupt).

Các mô hình điều khiển loại này được điều khiển bằng các sự kiện sinh ra từ bên ngoài. Dưới đây xét hai loại mô hình tiêu biểu trong số đó:

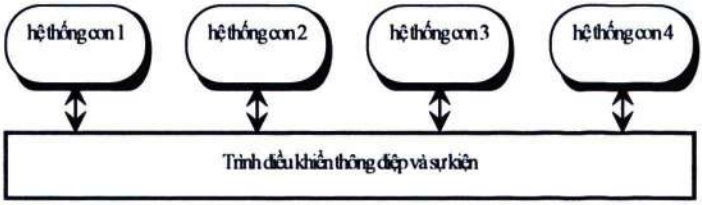
- *Mô hình điều khiển quảng bá (Broadcast model):* Trong các mô hình này, một sự kiện về nguyên tắc sẽ chuyển tới tất cả các hệ thống con mà có thể được đáp ứng lại sự kiện đó.

- *Mô hình điều khiển ngắt (Interrupt-driven model):* Mô hình này được dùng trong các hệ thống thời gian thực, các ngắt được một bộ điều khiển ngắt nhận ra. Sau đó chúng được chuyển đến một số thành phần khác để xử lý.

5.2.3.1. Mô hình điều khiển quảng bá

Trong một mô hình quảng bá, các hệ thống con đăng ký một vài sự kiện cụ thể mà chúng quan tâm. Khi một sự kiện xảy ra, điều khiển được chuyển tới hệ thống con đăng ký loại sự kiện này. Hệ con được truyền điều khiển khi đã có đăng ký. Mô hình này khá hiệu quả trong việc tích hợp các hệ con. Tuy vậy, hệ con không nhớ sự kiện đã được xử lý chưa.

Ưu điểm của các tiếp cận quảng bá là sự giải quyết các mối quan hệ một cách đơn giản. Một hệ thống con mới điều khiển những lớp cụ thể của các sự kiện có thể được tích hợp bằng cách đăng ký những sự kiện của nó với bộ điều khiển sự kiện. Nhược điểm của mô hình này là các hệ thống con không biết khi nào sự kiện được điều khiển. Khi một hệ thống con phát sinh ra một sự kiện, nó không biết những hệ thống con nào đã đăng ký sự kiện đó. Điều này có thể là nguyên nhân gây ra những xung đột khi những kết quả của việc điều khiển sự kiện đã có sẵn.

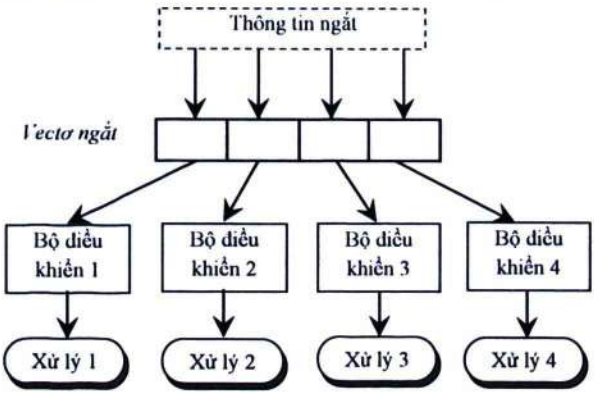


Hình 5.11. Một mô hình điều khiển dựa trên sự quảng bá có tuyển chọn

5.2.3.2. Mô hình điều khiển hướng ngắt

Các hệ thống thời gian thực mà yêu cầu các sự kiện sinh ra từ bên ngoài là hệ thống điều khiển bởi sự kiện. Mô hình điều khiển hướng ngắt được dùng trong các hệ thời gian thực. Bộ điều khiển ngắt nhận sự kiện và truyền đến các thành phần xử lý. Đây là mô hình có độ phản hồi nhanh. Tuy vậy, sử dụng mô hình này thì khó lập trình và đánh giá.

Một mô hình điều khiển ngắt được minh họa như trong hình 5.12 Bộ điều khiển ngắt được xác định cho một số kiểu ngắt. Mỗi kiểu ngắt được kết hợp với một vùng nhớ lưu trữ địa chỉ của bộ điều khiển. Khi một kiểu ngắt cụ thể nhận được, bộ chuyển mạch phần cứng tạo ra điều khiển để truyền tức thời sự kiện đó tới bộ điều khiển ngắt. Bộ điều khiển ngắt sẽ khởi động hoặc dừng các tiến trình khác nhau để đáp ứng yêu cầu của sự kiện ngắt nhận được.



Hình 5.12. Minh họa một mô hình điều khiển ngắt

Mô hình này chỉ nên được sử dụng trong các hệ thống thời gian thực phức tạp mà ở đó sự đáp ứng tức thời sự kiện là cần thiết. Nó có thể kết hợp với mô hình điều khiển tập trung. Những bộ điều khiển tập trung thường vận hành hệ thống với điều khiển dựa trên ngắt cho các trường hợp khẩn cấp.

5.2.4. Kiến trúc phần mềm theo mô hình phân rã mô-đun

Mô hình phân rã mô-đun là cấp độ kiến trúc mà các hệ thống con được phân rã thành các mô-đun.

Mô hình phân rã mô-đun bao gồm:

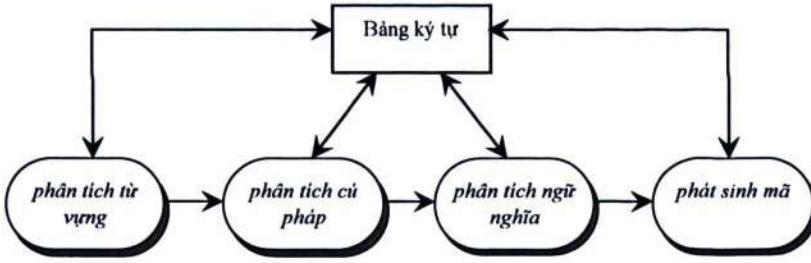
1- Mô hình đối tượng: hệ thống được phân ra thành tương tác đối tượng (chúng ta sẽ phân tích chi tiết ở chương sau);

2- Mô hình luồng dữ liệu: hệ thống được phân tách thành các mô-đun chức năng chuyển các đầu vào thành các đầu ra (mô hình *đường ống - pipeline*). Chúng ta sẽ trình bày chi tiết về mô hình này trong mục "5.4.Tiến trình thiết kế kiến trúc" ở phần sau.

5.2.5. Kiến trúc phần mềm theo miền ứng dụng

Những mô hình kiến trúc ở trên là những mô hình tổng quát. Ngoài ra, có những mô hình kiến trúc được xây dựng cho một miền ứng dụng cụ thể. Mặc dù các ứng dụng của một miền khác nhau rất nhiều về chi tiết, nhưng kiến trúc chung có thể được dùng lại khi phát triển những hệ thống mới. Những mô hình kiến trúc loại này được gọi là những *kiến trúc miền cụ thể (domain specific architecture)*. Có hai kiểu mô hình kiến trúc xác định cho miền:

1- Mô hình tổng quát (generic model) là sự khái quát hóa từ một số lượng lớn các hệ thống thực cụ thể cùng kiểu. Ví dụ, trong những hệ thống thời gian thực, có những mô hình kiến trúc chung cho những kiểu hệ thống khác nhau như các *hệ thu thập dữ liệu*, các *hệ thống theo dõi*,... Mô hình tổng quát phản ánh kiến trúc của những hệ thống đang tồn tại và có thể dùng lại một cách trực tiếp trong một thiết kế.



Hình 5.13. Mô hình luồng dữ liệu của một trình biên dịch

Trong mô hình trình biên dịch, hàng ngàn trình biên dịch đã được tạo ra theo mô hình này. Nó thường bao gồm một số mô-đun và có thể được tổ chức

theo những mô hình kiến trúc khác nhau về chi tiết. Các pha phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa được tổ chức tuần tự.

2- Kiến trúc tham chiếu (reference architectural model) có mức trừu tượng cao hơn và khái quát hóa một số lớn các hệ thống. Nó cung cấp những thông tin về kiến trúc như những chuẩn để so sánh. Những mô hình tham chiếu thường nhận được từ một sự học tập, khái quát kinh nghiệm của miền ứng dụng. Chúng mô tả một kiến trúc lý tưởng hóa bao gồm tất cả những tính năng mà hệ thống có thể có. Nó được dùng như một cơ sở cho sự thực thi hệ thống. Điều này chính là mục đích của kiến trúc OSI (Open System Interconnection), do Zimmermann đề xuất năm 1980 [23], dùng cho việc liên kết các hệ thống mở. Nó dự kiến xây dựng như một chuẩn. Nếu một hệ thống phù hợp với mô hình này thì có thể giao tiếp với các hệ thống khác phù hợp với nó.

Hình 5.14. Kiến trúc của mô hình tham chiếu OSI

5.3. Một số phương pháp tạo kiến trúc

Có hai phương pháp chính để tạo kiến trúc. Đó là:

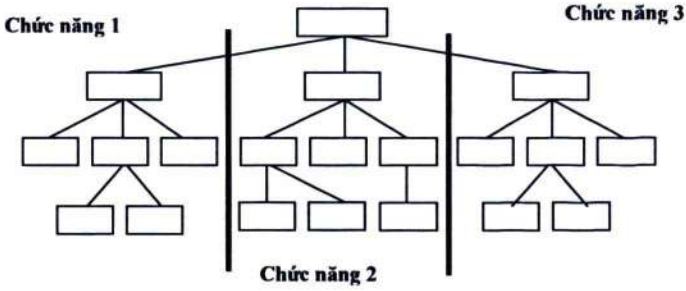
1- Phân hoạch kiến trúc (bao gồm phân hoạch ngang và phân hoạch dọc);

2- Phương pháp tạo kiến trúc từ biểu đồ luồng dữ liệu.

5.3.1. Phân hoạch ngang

Phân hoạch ngang xác định các nhánh rẽ riêng biệt cho các chức năng chủ chốt và sử dụng các mô-đun điều khiển để điều phối thông tin giữa các chức năng.

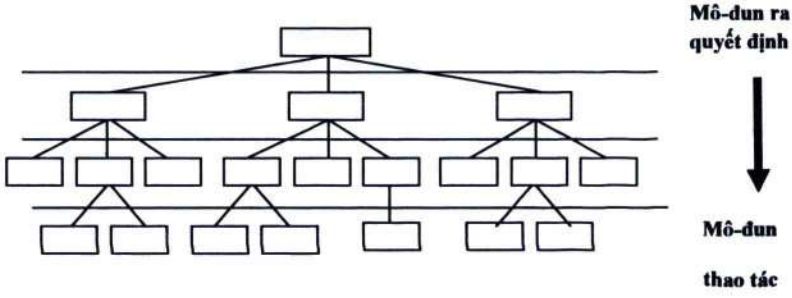
Từ sơ đồ phân cấp chức năng, người ta bổ dọc sơ đồ này để có được một phân hoạch như hình 5.15 sau:



Hình 5.15. Phân hoạch ngang các chức năng

5.3.2. Phân hoạch dọc

Phân hoạch dọc thực hiện phân tầng: mô-đun điều khiển (ra quyết định) và mô-đun thao tác (workers). Mô-đun ra quyết định được xếp ở tầng cao như hình 5.16 sau:



Hình 5.16. Phân hoạch dọc các chức năng

Cả hai kiểu phân hoạch trên đều dễ kiểm thử, bảo trì, dễ mở rộng nhưng có hạn chế là có thể có hiệu ứng phụ khi sửa đổi.

5.3.3. Phương pháp tạo kiến trúc từ biểu đồ luồng dữ liệu

Phương pháp chính để tạo kiến trúc là phương pháp tạo kiến trúc từ biểu đồ luồng dữ liệu. Phương pháp này sẽ được trình bày chi tiết trong các phần ở sau đây.

5.4.Tiến trình thiết kế kiến trúc hệ thống phần mềm

5.4.1. Những hoạt động chung cho mọi tiến trình thiết kế

Kiến trúc phần mềm được thể hiện qua hai đặc trưng quan trọng của chương trình máy tính là cấu trúc dữ liệu và cấu trúc cấp bậc điều khiển của các mô-đun, nhưng nội dung thiết kế kiến trúc tập trung vào việc *biểu diễn cấu trúc các thành phần phần mềm, các thuộc tính của nó và tương tác giữa chúng*. Các nhà thiết kế khác nhau có thể tiếp cận tiến trình thiết kế kiến trúc theo cách khác nhau. Tuy nhiên, những hoạt động sau đây là chung cho mọi tiến trình [16]:

1- *Cấu trúc hệ thống:* Một hệ thống được cấu trúc thành một số hệ thống con.

2- *Mô hình hóa điều khiển*: Thiết lập một mô hình chung mô tả mối quan hệ điều khiển giữa các phần trong hệ thống.

3- *Phân rã các mô-đun*: Mỗi hệ thống con được phân rã thành các mô-đun. Kiến trúc này cần chỉ ra các loại mô-đun và liên kết giữa chúng.

Giai đoạn đầu tiên của tiến trình thiết kế kiến trúc là phân rã một hệ thống thành một tập các hệ thống con tương tác với nhau. Ở mức độ trừu tượng nhất, kiến trúc hệ thống được mô tả như một biểu đồ khối bao gồm các hệ thống con. Các hệ thống con này lại có thể phân thành các hệ thống con của nó.

Trên thực tế, khi sử dụng các tiến trình phát triển phần mềm mang tính hệ thống cao (như *RUP* hay *tiến trình phát triển hướng cấu trúc* [16]), việc cấu trúc mô hình kiến trúc của hệ thống được thực hiện từ chính các bước phát triển của các tiến trình đó.

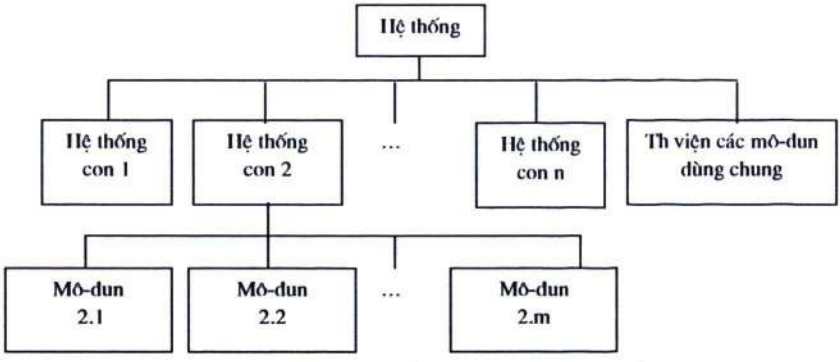
Sau khi một kiến trúc mức khái quát đã được thiết kế, giai đoạn tiếp theo của tiến trình thiết kế kiến trúc là phân rã các hệ con thành mô-đun. Các cấp độ khác nhau của sự phân rã có thể là một phần của tiến trình thiết kế kiến trúc.

Phần trên, chúng ta đã giới thiệu hai mô hình thường được sử dụng khi phân rã một hệ thống con thành các mô-đun là *mô hình hướng đối tượng (object-oriented model)* và *mô hình luồng dữ liệu (data-flow model)*. Từ đây, chúng ta sẽ tập trung nghiên cứu về mô hình luồng dữ liệu. Khi đó, hệ thống được phân rã thành các mô-đun chức năng cho phép chuyển đổi dữ liệu vào

thành dữ liệu ra của nó. Phương thức này cũng được gọi là một *cách tiếp cận đường ống (pipeline)*. Chú ý rằng cách tiếp cận này không phù hợp với hệ thống tương tác.

Mỗi mô-đun có thể tương ứng với một hoặc một phần hoặc một vài chức năng hoặc tiến trình của hệ thống. Mỗi chức năng là một tiến trình chuyển hóa đầu vào thành đầu ra.

Mỗi mô-đun có thể chỉ được dùng ở một chương trình, có mô-đun được dùng ở nhiều chương trình. Những mô-đun dùng chung thường được cất giữ riêng ở một chỗ, thường được gọi là thư viện các mô-đun hay các chương trình mẫu. Nhiều chương trình mẫu có thể sử dụng cho nhiều hệ thống con khác nhau, tức là có thể sử dụng lại được. Chúng ta có sơ đồ cấu trúc tĩnh của một hệ thống như hình 5.17.



Hình 5.17. Mô hình kiến trúc tĩnh của hệ thống

5.4.2. Quy trình thiết kế kiến trúc chương trình

Về mặt lý thuyết, có hai công nghệ xây dựng kiến trúc hệ thống [9]. Về mặt thực hành, việc xây dựng lược đồ chương trình thường tuân theo quy trình hai bước như sau [10]:

Bước 1- Xây dựng mô hình luồng dữ liệu hệ thống;

Bước 2- Xây dựng lược đồ chương trình.

Đầu ra của thiết kế chương trình là một đặc tả về nội dung các chương trình sẽ được cài đặt, bao gồm:

+ Một lược đồ chương trình cho mỗi hệ thống con. Lược đồ chương trình được đặc tả dưới dạng đồ thị có hướng trong đó mỗi nút là một mô-đun chương trình, mỗi cung là một lời gọi (lời gọi của mô-đun ở gốc đối với mô-đun ở ngọn của cung);

+ Đặc tả nội dung của từng mô-đun trong lược đồ chương trình;

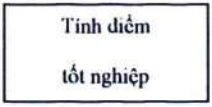
+ Phân bổ các mô-đun trong lược đồ chương trình thành các chương trình (hay mô-đun tải).

Như vậy, lược đồ chương trình chính là một biểu diễn dưới dạng đồ thị của một tập hợp các mô-đun cùng với các giao diện giữa các mô-đun đó (bao gồm sự chuyển giao điều khiển và chuyển giao dữ liệu).

Các yếu tố cấu thành lược đồ chương trình bao gồm [1]:

1- *Các mô-đun*: biểu diễn bởi hình chữ nhật có tên mô-đun ở trong. Tên mô-đun phản ánh tóm tắt chức năng của nó.

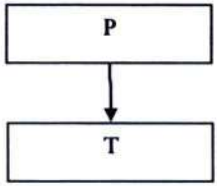
Ví dụ 5.2:



2- *Kết nối các mô-đun:*

Các mô-đun có thể được kết nối với nhau bằng các lời gọi, diễn tả bằng một mũi tên (cung):

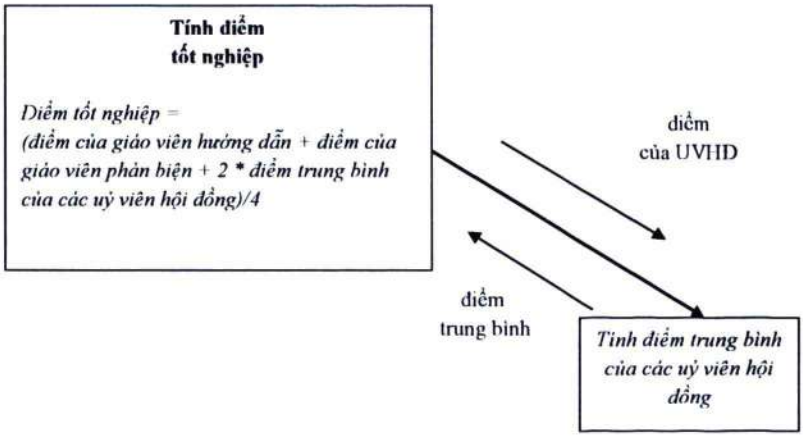
- Mô-đun P gọi mô-đun T
- Mô-đun T thực hiện chức năng của mình rồi trả điều khiển cho P ở vị trí sau lời gọi.



3- Thông tin trao đổi giữa các mô-đun:

Các thông tin được gửi đi kèm với lời gọi (các tham số) và thông tin trả về sau khi thực hiện lời gọi (trả lời) được thể hiện bằng các mũi tên nhỏ vẽ dọc theo cung biểu diễn cho lời gọi, có kèm theo tên của thông tin.

Ví dụ 5.3:



Hình 5.18. Minh họa các yếu tố cấu thành lược đồ cấu trúc chương trình

5.5. Phân tích đặc tả yêu cầu phần mềm thành đặc tả kiến trúc phần mềm

Có một số kỹ nghệ điển hình dùng để phân tích đặc tả yêu cầu phần mềm thành đặc tả kiến trúc phần mềm như hướng luồng dữ liệu, hướng đối tượng hay hướng dữ liệu,... Mỗi kỹ nghệ phân tích đều có những ưu điểm riêng của nó, trong đó kỹ nghệ "*hướng luồng dữ liệu*" được sử dụng phổ cập hơn cả.

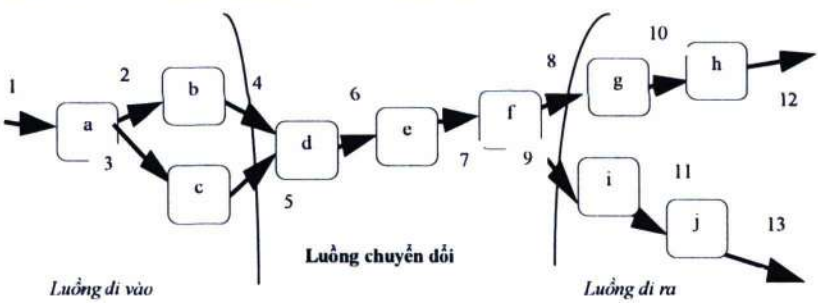
Mục tiêu của kỹ nghệ thiết kế theo hướng luồng dữ liệu là cung cấp một cách tiếp cận có hệ thống cho việc suy dẫn ra cấu trúc chương trình, cung cấp một cách nhìn toàn cục về phần mềm và cột chống cho thiết kế kiến trúc.

Stevens, Myers và Constantine [23] đã trình bày thiết kế phần mềm dựa trên sự chuyển đổi luồng dữ liệu thành cấu trúc hệ thống mô-đun. Chúng ta biết rằng thông tin có thể biểu diễn như một luồng liên tục chảy ngầm dưới một loạt các phép biến đổi (tiến trình) khi nó tiến hóa từ cái vào thành cái ra. Biểu diễn luồng thông tin là một phần của phân tích yêu cầu, đồng thời được gọi là phân tích miền thông tin. Biểu đồ luồng dữ liệu được dùng như một công cụ đồ họa để mô tả luồng thông tin. Thiết kế hướng luồng dữ liệu xác định ra một số ánh xạ khác nhau làm biến đổi luồng thông tin thành cấu trúc chương trình.

Chú ý rằng thiết kế hướng luồng dữ liệu thường được gọi là "*thiết kế có cấu trúc*" có nguồn gốc từ những khái niệm ban đầu nhấn mạnh vào tính mô-đun, thiết kế trên xuống và lập trình có cấu trúc. Phương pháp thiết kế này đã và đang được áp dụng cho một phạm vi rộng các miền ứng dụng. Cách tiếp cận hướng luồng dữ liệu đối với thiết kế đặc biệt có ích khi thông tin được xử lý tuần tự và không tồn tại cấu trúc dữ liệu cấp bậc hình thức (cấu trúc dữ liệu có thể biểu diễn bằng sơ đồ phân cấp và được đặc tả chính xác nhờ việc sử dụng công cụ chủ yếu là đại số và logic, không diễn đạt bằng ngôn ngữ thông thường). Tuy nhiên, cũng có trường hợp mà việc xem xét luồng dữ liệu cũng chỉ là một vấn đề phụ, khi đó thì một số phương pháp khác như hướng đối tượng hay hướng dữ liệu lại có thể thích hợp hơn.

5.5.1. Khái niệm luồng chuyển đổi và luồng giao dịch

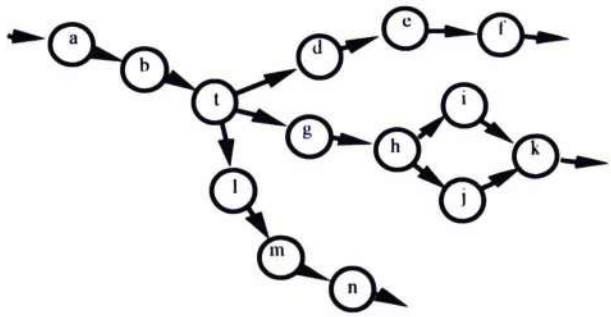
Thông tin đi vào hệ thống theo con đường biến đổi dữ liệu ngoài thành dạng bên trong và được coi như luồng đi vào. Trong hạt nhân của phần mềm, việc chuyển đổi xuất hiện. Dữ liệu tới được truyền qua một trung tâm chuyển đổi và bắt đầu chuyển dọc theo con đường bây giờ dẫn tới "cái ra" của phần mềm. Dữ liệu chuyển dọc theo những con đường này được gọi là luồng đi ra. Toàn bộ luồng dữ liệu xuất hiện theo một cách tuần tự đi theo một, hay một vài, đường "trực tiếp". Khi một đoạn của biểu đồ luồng dữ liệu có những đặc trưng này thì đó là luồng chuyển đổi (Transform Flow).



Hình 5.19. Ví dụ minh họa luồng chuyển đổi

Tuy nhiên, luồng thông tin có thể được đặc trưng bởi một khoản mục dữ liệu riêng biệt, gọi là một giao dịch (hay còn gọi là giao tác), gây ra luồng dữ

liệu khác đi theo một trong nhiều đường dẫn. Khi một luồng dữ liệu lấy dạng như chỉ ra trong hình 5.20, luồng giao dịch (Transaction Flow) được xác định.



Hình 5.20. Ví dụ minh họa luồng giao dịch

Luồng giao dịch được đặc trưng bởi dữ liệu chuyển theo đường đi vào (cũng còn được gọi là đường nhận) đối thông tin thế giới ngoài thành một giao dịch. Giao dịch được tính toán và dựa trên giá trị của nó, luồng được khởi đầu theo một trong nhiều đường hành động. Đầu chia của luồng thông tin mà từ đó nhiều đường hành động đi ra còn được gọi là *trung tâm giao dịch*.

Một HTTT hay những phần của nó thường là một hệ thống tập trung giao dịch (transaction centered system) hoặc là một *hệ thống tập trung chuyển đổi (transform centered system)*. Một hệ thống tập trung chuyển đổi có chức năng cơ bản là tạo ra các dữ liệu mới từ các dữ liệu có được. Ví dụ như hệ thống tính điểm trung bình học kỳ cho sinh viên (điểm trung bình học kỳ được tính trên cơ sở đã có các điểm môn học trong kỳ đó). Còn trong hệ thống giao dịch tập trung, chức năng cơ bản của nó là gửi các dữ liệu được nhận dạng tới một vị trí cụ thể trong hệ thống để xử lý. Các dữ liệu đi vào một mô-đun trung tâm giao dịch của hệ thống và chúng được gửi đi tới các vị trí cụ thể tùy theo loại dữ liệu. Một ví dụ về hệ thống tập trung giao dịch là một hệ thống ngân hàng dược thiết kế để xử lý các giao dịch (xử lý tiền gửi, xử lý thanh toán, xử lý rút vốn) về ngân hàng,

Trong hệ thống lớn, cả hai luồng chuyển đổi và giao dịch đều có thể hiện hữu trong biểu đồ luồng dữ liệu. Chẳng hạn, trong luồng giao dịch, luồng thông tin theo một đường hành động có thể có các đặc trưng của luồng chuyển đổi.

5.5.2. Phân tích chuyển đổi thành kiến trúc phần mềm

Phân tích chuyển đổi (Transform Analysis) là một tập các bước thiết kế cho phép một luồng dữ liệu với các đặc trưng luồng chuyển đổi được ánh xạ vào một tiêu bản đặt sẵn cho cấu trúc chương trình [23]. Đây không phải là một thuật toán mà là một quá trình cấu trúc chưa hoàn toàn xác định. Mỗi lần chọn một biểu đồ luồng dữ liệu ở một mức nào đó để chuyển sang biểu đồ cấu trúc có thể dẫn đến kết quả không giống nhau.

Quá trình thiết kế kiến trúc có thể phân tích thành nhiều bước:

Bước1-Duyệt xét lại mô hình hệ thống nền tảng: mô hình hệ thống nền tảng bao gồm luồng dữ liệu các mức ngữ cảnh và mức đỉnh, và các thông tin hỗ trợ. Trong thực tế, bước thiết kế được bắt đầu với việc đánh giá cả bản đặc tả hệ thống và đặc tả yêu cầu phần mềm. Cả hai tài liệu này đều mô tả luồng và cấu trúc thông tin tại giao diện phần mềm.

Bước 2-Xét duyệt và làm mịn các biểu đồ luồng dữ liệu cho phần mềm: thông tin thu được từ các mô hình phân tích có trong bản đặc tả yêu cầu phần mềm sẽ được làm mịn để tạo ra chi tiết nhiều hơn. Chúng ta cần chi tiết hóa đến mức nào mà mỗi biến đổi trong biểu đồ luồng dữ liệu có sự kết dính tương đối cao. Tức là, tiến trình được bao hàm bởi phép biến đổi sẽ thực hiện một chức năng riêng biệt mà có thể được cài đặt như một mô-đun trong phần mềm đó.

Bước 3- Xác định xem liệu luồng dữ liệu có đặc trưng chuyển đổi hay đặc trưng giao dịch bằng việc nhận diện có trung tâm chuyển đổi hay có trung tâm giao dịch không. Nếu đúng cho trung tâm chuyển đổi thì chúng ta sẽ tiếp sang bước sau.

Bước 4- Cô lập trung tâm chuyển đổi bằng cách xác định các đường biên xác định luồng đi vào và luồng đi ra.

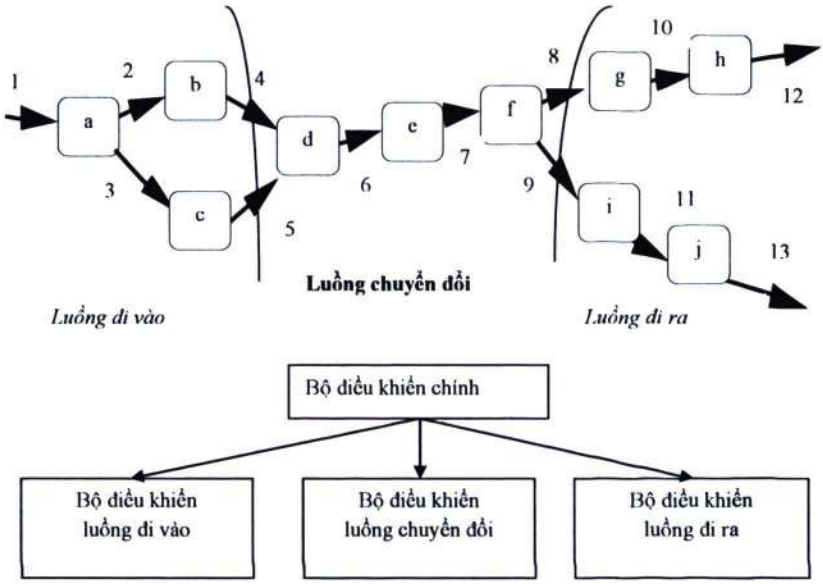
Bước 5- Xây dựng mô hình kiến trúc mức khái quát.

Cấu trúc chương trình biểu diễn một phân bố điều khiển từ trên xuống. Việc lấy một mô-đun điều khiển chung trong chương trình náy sinh trong một cấu trúc chương trình, trong đó các mô-đun mức đỉnh thực hiện việc ra quyết định còn các mô-đun mức thấp thì thực hiện hầu hết các công tác vào, xử lý và

đưa ra. Các mô-đun mức trung thực hiện một số điều khiển và tiến hành một khối lượng vừa phải công việc.

Bộ điều khiển chính (thường được gọi là Main Program) nằm tại đỉnh của cấu trúc chương trình và điều phối các chức năng điều khiển thuộc cấp dưới. Bộ điều khiển xử lý thông tin đi vào, phối hợp việc nhận mọi dữ liệu tới. Bộ điều khiển luồng chuyển đổi, giám sát mọi thao tác trên dữ liệu dưới dạng trung gian (như một mô-đun gọi đến nhiều thủ tục biến đổi dữ liệu).

Bộ điều khiển xử lý thông tin đi ra, phối hợp việc tạo thông tin ra.



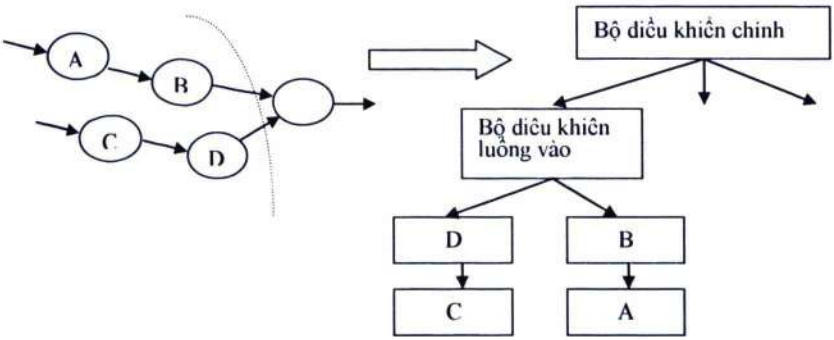
Hình 5.21. Minh họa mô hình kiến trúc mức khái quát

Chú ý rằng, từng mô-đun điều khiển trong mô hình kiến trúc đều được cho một cái tên bao hàm chức năng của các mô-đun thuộc cấp mà nó kiểm soát.

Bước 6- Phát triển mô hình kiến trúc mức chi tiết

Ở mức chi tiết, chúng ta ánh xạ các phép biến đổi riêng (hình tròn) của luồng dữ liệu vào các mô-đun thích hợp bên trong cấu trúc chương trình. Bắt đầu từ luồng chuyển đổi, sau đó đến luồng đi vào và luồng đi ra. Các phép biến đổi được ánh xạ vào các mức thuộc cấp của cấu trúc chương trình.

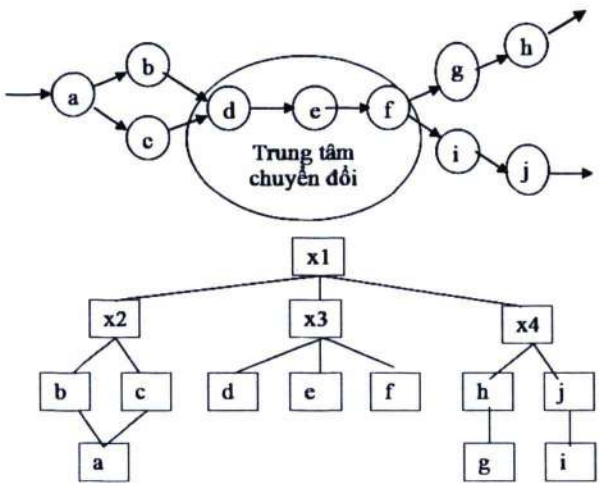
Ví dụ 5.4:



Hình 5.22. Phát triển mô hình cấu trúc mức chi tiết

Chú ý rằng, có thể tổ hợp nhiều tiến trình thành một mô-đun hoặc ngược lại, một tiến trình phân rã thành một vài mô-đun tùy thuộc vào bài toán thực tế miễn là đảm bảo các yếu tố chất lượng (tính kết dính, tính ghép nối và tính hình thái).

Các mô-đun được ánh xạ theo cách thức mô tả trên sẽ tạo ra được mô hình kiến trúc biểu thị cho một thiết kế ban đầu của cấu trúc chương trình.



Hình 5.23. Ví dụ phân tích luồng chuyển đổi

Bước 7- Làm mịn cấu trúc chương trình bằng cách dùng trực cảm thiết kế để cải thiện chất lượng phần mềm:

Việc làm mịn cấu trúc chương trình được tiến hành bằng cách áp dụng các khái niệm độc lập mô-đun và xem xét hoàn cảnh thực tế. Đối với các nhóm mô-đun không quá phức tạp có thể không cần đến mô-đun điều khiển cấp trung gian. Xem xét lại những mô-đun nào có thể lồng vào nhau để giảm bớt độ ghép nối. Không để mô-đun quá to, quá phức tạp, mô-đun quá nhỏ hoặc quá đơn giản.

Mục tiêu của bẩy bước nói trên là để phát triển một biểu diễn toàn cục cho phần mềm. Tức là một khi cấu trúc được xác định thì chúng ta có thể đánh giá và làm mịn kiến trúc phần mềm bằng cách coi nó như một tổng thể. Những sửa đổi vào lúc này ít đòi hỏi thêm công việc nhưng có thể có tác động sâu sắc lên chất lượng và việc bảo trì phần mềm.

5.5.3. Phân tích giao dịch thành kiến trúc phần mềm

Trong nhiều ứng dụng phần mềm, một khoản mục dữ liệu phát động một hay một số luồng thông tin ảnh hưởng tới chức năng được ngụ ý bởi khoản mục dữ liệu gây ra đó. Khoản mục dữ liệu này được gọi là giao dịch (hay giao tác), và các đạc trưng luồng tương ứng của nó đã được xét đến ở phần trước.

Các bước thiết kế cho phân tích giao dịch là tương tự nhau và trong một số trường hợp đồng nhất với các bước cho phân tích chuyển đổi. Điều khác biệt chính nằm ở chỗ ánh xạ luồng dữ liệu vào cấu trúc chương trình.

Bước 1- Xét duyệt mô hình hệ thống nền tảng: Các hoạt động của bước này giống như phân tích chuyển đổi

Bước 2- Xét duyệt và làm mịn biểu đồ luồng dữ liệu cho phần mềm

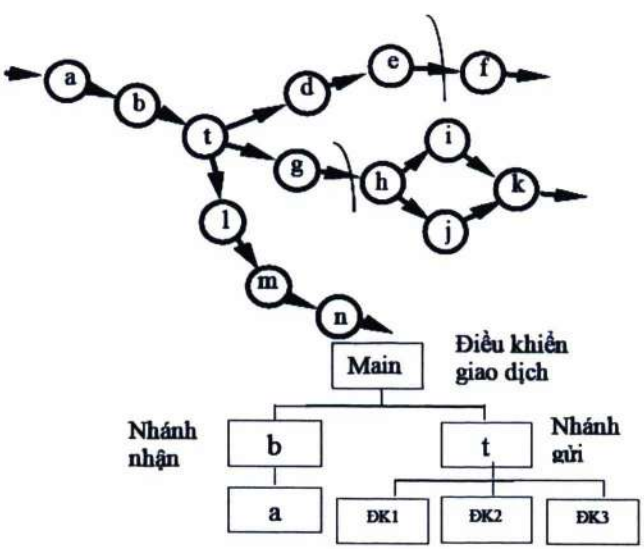
Bước 3- Xác định xem liệu luồng dữ liệu có đặc trưng luồng chuyển dịch hay giao dịch không. Nếu có trung tâm giao dịch thì chúng ta chuyển sang bước sau.

Bước 4- Xác định trung tâm giao dịch và các đặc trưng luồng theo từng đường hành động: Vị trí của trung tâm giao dịch có thể được nhận ra ngay từ sơ đồ luồng dữ liệu. Trung tâm giao dịch nằm tại gốc của một số đường hành động.

Đường đi vào (tức là đường theo đó giao dịch được nhận) và mọi đường hành động cũng phải được cô lập. Các biên giới xác định ra đường nhận và đường hành động cần được chỉ rõ. Mỗi đường hành động phải được ước lượng cho đặc trưng luồng riêng.

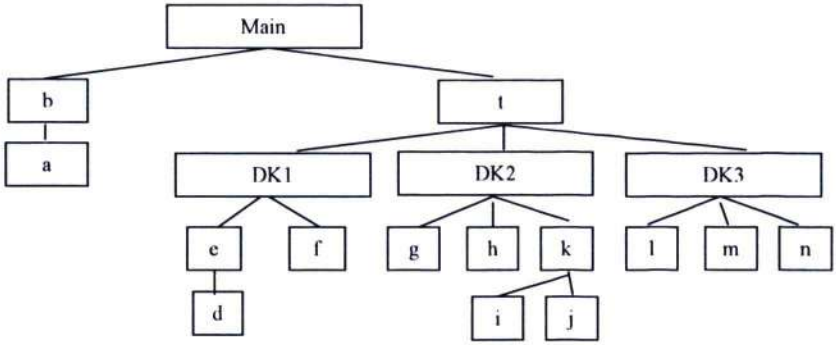
Bước 5- Ánh xạ luồng dữ liệu vào cấu trúc chương trình theo xử lý giao dịch (mức khái quát):

Luồng giao tác được ánh xạ vào cấu trúc chương trình có chứa một nhánh đi vào và một nhánh gửi. Cấu trúc cho nhánh đi vào được phân tích theo cùng cách như phân tích chuyển đổi. Bắt đầu từ trung tâm giao dịch, các hình tròn trên đường đi vào (đường nhận) được ánh xạ vào các mô-đun. Cấu trúc của nhánh gửi có chứa một mô-đun gửi kiểm soát mọi bộ điều khiển đường hành động thuộc cấp. Tiến trình này được minh họa trên hình 5.24.



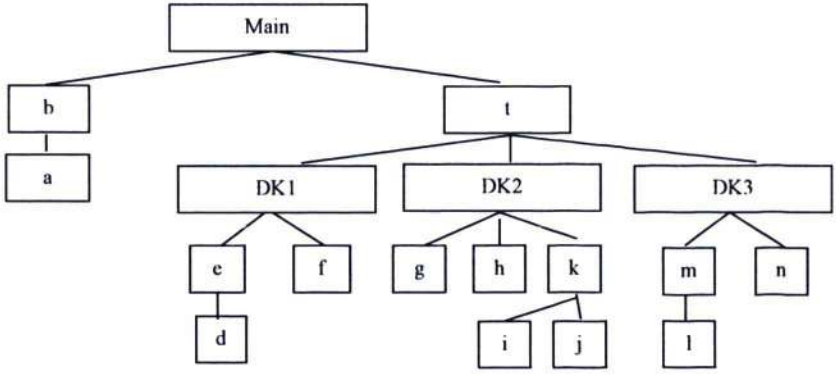
Hình 5.24. Ánh xạ giao dịch

Bước 6- Ánh xạ các đường hành động vào trong sơ đồ cấu trúc (mức chi tiết): Mỗi đường hành động của sơ đồ luồng dữ liệu được ánh xạ vào một cấu trúc tương ứng với các đặc trưng luồng riêng.



Hình 5.25. Cấu trúc chương trình phác thảo

Bước 7- Làm mịn cấu trúc chương trình bằng trực cảm thiết kế để nâng cao chất lượng phần mềm: Bước này đối với phân tích giao dịch là đồng nhất với bước tương ứng cho phân tích chuyển đổi.



Hình 5.26. Cấu trúc chương trình được làm mịn

Trong cả hai cách tiếp cận thiết kế trên, các tiêu chuẩn như độc lập mô-đun, tính thực tế (hiệu quả của cài đặt và kiểm thử) và tính bảo trì phải được xem xét cẩn thận như những thay đổi cấu trúc đề nghị.

5.6. Chất lượng thiết kế kiến trúc phần mềm

5.6.1. Một số tiêu chuẩn thiết kế kiến trúc phần mềm

Mục tiêu của một thiết kế tốt nói chung và thiết kế kiến trúc nói riêng là tạo ra một hệ thống "thân thiện với người dùng". Đó là một hệ thống dễ đọc, dễ

chuyển thành chương trình và dễ bảo trì. Muốn thế, kiến trúc hệ thống cần rõ ràng, các hệ con phải độc lập tương đối với nhau, tức là gắn kết với nhau lỏng lẻo. Các hệ con nên chỉ kết nối với nhau qua giao diện. Nhờ vậy có thể phát triển các hệ con độc lập với nhau. Các mô-đun cần được phân đủ nhỏ thành các đơn vị chức năng riêng biệt và gắn kết với nhau chặt chẽ theo từng chức năng. Các thành phần phần mềm cần độc lập tương đối và liên kết lỏng lẻo để dễ thích nghi; và việc sửa đổi một thành phần sẽ ít ảnh hưởng đến các thành phần khác. Điều này dẫn đến hai đặc trưng quan trọng mà một thiết kế cần có là "sự kết dính" giữa các bộ phận trong một mô-đun chức năng phải chặt chẽ và "sự ghép nối" giữa các thành phần trong hệ thống phải lỏng lẻo.

5.6.1.1. Sự kết dính

Sự kết dính (Cohesion) của một thành phần là độ đo về tính gắn kết chặt chẽ với nhau giữa các bộ phận của nó. Nói khác đi, sự kết dính nói lên sự gắn bó giữa các phần bên trong một mô-đun. Muốn vậy, một thành phần (mô-đun) phải thực hiện một chức năng logic hoặc một thực thể logic, và tất cả các bộ phận của nó đều phải tham gia vào việc thực hiện này. Nếu một phần không trực tiếp tham gia vào việc thực hiện chức năng logic đó thì mức độ kết dính của nó là thấp.

Page-Jones (1980) đưa ra bảy mức kết dính theo thứ tự kết dính tăng dần sau đây:

1- Kết dính gom góp: Các phần của một thành phần không liên quan với nhau, song lại bị bó vào một thành phần.

2- Kết dính hội hợp logic: Các thành phần cùng thực hiện các chức năng tương tự (xử lý lỗi, cập nhật,...) được đặt vào cùng một thành phần.

3- Kết dính theo thời điểm: Tất cả các thành phần cùng hoạt hóa một lúc (cùng bắt đầu, cùng kết thúc,...) được ghép lại với nhau.

4- *Kết dính thủ tục:* Các phần tử trong thành phần được ghép lại trong một dãy điều khiển.

5- *Kết dính truyền thông*: Tất cả các phần tử của thành phần cùng thao tác trên một dữ liệu vào và đưa ra cùng một dữ liệu ra.

6- *Kết dính tuần tự*: Trong một thành phần, cái ra của phần tử này là cái vào của phần tử khác.

7- *Kết dính chức năng*: Mỗi phần của thành phần đều là cần thiết để thực hiện cùng một chức năng nào đó.

Các lớp kết dính trên đây không được định nghĩa chặt chẽ và cũng không phải dễ xác định. Một đối tượng kết dính thể hiện ra như một thực thể đơn, tức là mọi phép toán mà nó thực hiện đều nằm trong đó. Vậy có một lớp kết dính nữa là:

8- Kết dính đối tượng: Mỗi phép toán thực hiện một chức năng. Các chức năng này tác động chỉ lên các thuộc tính của đối tượng để cải biên, thanh tra và sử dụng chúng làm cơ sở cho việc cung cấp dịch vụ của đối tượng.

5.6.1.2. Sự ghép nối

Ghép nối (coupling) chỉ ra mức độ độc lập giữa các đơn vị thành phần của một chương trình. Hệ thống có ghép nối cao sẽ có độ liên kết mạnh giữa các thành phần, làm chúng phụ thuộc lẫn nhau nhiều hơn. Hệ thống ghép nối lỏng lẻo làm cho các thành phần trở nên độc lập hoặc tương đối độc lập với nhau.

Sự ghép nối của một thành phần với các thành phần khác thể hiện ra bằng *số lượng mối liên kết* phụ thuộc giữa nó với các thành phần khác, cũng như bản chất của các mối quan hệ đó. Chẳng hạn, mối quan hệ đồng bộ (gọi chờ đáp lại) là chặt hơn mối quan hệ không đồng bộ (gửi thông báo đi). Thừa kế trong hệ thống hướng đối tượng dẫn tới một dạng ghép nối chặt giữa các lớp có quan hệ kế thừa. Việc tối thiểu hoá sự ghép nối không phải luôn luôn có thể thực hiện được.

Page-Jones (1980) đã giới thiệu năm loại ghép nối và trình bày ở đây theo thứ tự từ tốt đến xấu:

1- *Ghép nối dữ liệu:* Các mô-đun truyền thông với nhau bằng các phần tử dữ liệu hay thông tin điều khiển (cờ). Không một mô-đun nào có thể biết những gì diễn ra trong một mô-đun khác.

2- *Ghép nối nhãn*: Các dữ liệu được gửi đi là những cấu trúc dữ liệu hay toàn bộ một bản ghi. Sự thay đổi trong cấu trúc dữ liệu sẽ tác động lên mọi mô-đun sử dụng nó, chúng phụ thuộc nhau nhiều hơn.

3- *Ghép nối điều khiển*: Việc điều khiển xảy ra khi một mô-đun gửi đi *thông tin điều khiển* đến mô-đun khác. Thông tin này thể hiện như một cờ trong sơ đồ cấu trúc, nó thông báo cho mô-đun nhận cần phải làm cái gì.

4- *Ghép nối chung*: Khi hai mô-đun tham chiếu đến cùng một dữ liệu tổng thể, chúng được xem là ghép nối chung. Trong một số ngôn ngữ cho phép có vùng dữ liệu tổng thể đó thì lỗi từ vùng dữ liệu tổng thể có nguy cơ lan ra toàn hệ thống.

5- *Ghép nối nội dung*: Loại này xảy ra khi một mô-đun trực tiếp tham chiếu đến hoạt động của một mô-đun khác. Đây là loại ghép nối xấu nhất.

Các lớp kết dính và ghép nối không được định nghĩa chặt chẽ và cũng không phải luôn dễ xác định. Việc nhận diện chúng và vận dụng chúng trong thiết kế kiến trúc cần có kinh nghiệm thực tiễn.

5.6.1.3. Hình thái

Hình thái (morphology) của lược đồ cũng là một biểu hiện của chất lượng [1]. Khi đi từ mức trên xuống mức dưới, các nhánh của lược đồ chương trình có thể xòe ra (rẽ thành nhiều nhánh) hay chụm vào (chập vào cùng một mô-đun). Nhìn một cách toàn thể, lược đồ chương trình nên xòe ra ở trên và chụm lại ở dưới.

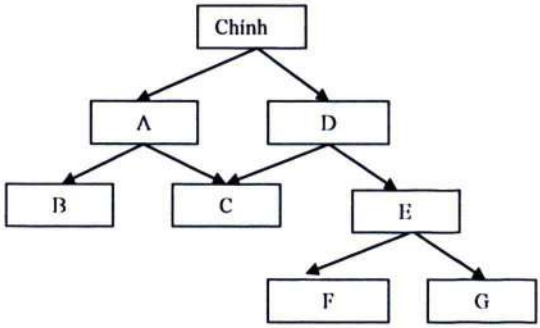
Thông thường thì trong một lược đồ chương trình, các mô-đun càng ở phía trên thì chức năng của chúng càng nặng về điều khiển và càng nhẹ về xử lý. Trái lại, các mô-đun càng ở thấp phía dưới thì càng ít tính điều khiển và càng nặng về xử lý hay dịch vụ. Bởi vậy càng xuất hiện nhiều điểm chụm ở phía dưới nghĩa là có nhiều dịch vụ được sử dụng đi sử dụng lại nhiều lần, ở nhiều chỗ. Sử dụng lại là một trong những mục tiêu thiết kế mô-đun, thể hiện trong lược đồ chương trình ở sự chụm lại đó.

Chúng ta gọi phạm vi điều khiển của một mô-đun là phần lược đồ chương trình bao gồm mô-đun đó và những mô-đun phụ thuộc (được gọi) trực

tiếp hay gián tiếp từ đó. Ta gọi phạm vi ảnh hưởng của một quyết định là phần lược đồ chương trình bao gồm mọi mô-đun chịu ảnh hưởng của quyết định đó.

Ví dụ 5.5:

Trong hình 5.27, phạm vi điều khiển của A là A, B, C còn phạm vi điều khiển của D là D, C, E, F, G. Nếu trong D có quyết định qd1 và kết quả của quyết định này được dùng trong C, E, F thì phạm vi ảnh hưởng của qd1 là C, E, F.



Hình 5.27. Minh họa một lược đồ chương trình

Một thiết kế tốt phải tạo ra một lược đồ chương trình, trong đó:

- Các quyết định phải có miền ảnh hưởng càng hẹp càng tốt;
- Mỗi phạm vi ảnh hưởng phải nằm trong phạm vi điều khiển tương ứng.

5.6.2. Các hướng dẫn thiết kế kiến trúc

Để có được một thiết kế tốt cần tuân thủ các nguyên tắc sau [17]:

- Hệ thống cần được mô-đun hóa, tức là nó cần được tổ chức thành một hệ phân cấp với một số đơn vị nhỏ hơn;
- Mỗi mô-đun cần kiểm soát được các chức năng cú một số hợp lý các mô-đun con ở mức thấp hơn;
- Các mô-đun cần độc lập tương đối với nhau, sao cho không một chức năng của mô-đun con nào có thể ảnh hưởng đến sự là việc bên trong của các mô-đun khác, tức là tổng số các truyền thông giữa các mô-đun cần phải đạt tối thiểu;

-Mỗi mô-đun cần có kích thước hợp lý;

-Cần làm mịn sao cho mỗi mô-đun chỉ thực hiện một và chỉ một chức năng;

-Mã hóa trong một mô-đun cần khái quát tới mức sao cho mỗi mô-đun có thể được sử dụng lại nhiều lần trong hệ thống.

Từ các nguyên tắc trên, người ta đã cụ thể hóa một số hướng dẫn thiết kế kiến trúc [17]:

1- Phân chia hệ thống thành các thành phần nhỏ hơn: chia nhỏ hệ thống thành các thành phần nhỏ hơn đến mức cần thiết;

2- Mở rộng kiểm tra: không một mô-đun nào kiểm soát quá 7 mô-đun con;

3- Ghép nối: tối thiểu hóa việc mở rộng các mô-đun đã là độc lập với mô-đun khác. Làm như vậy để tối thiểu hóa truyền thông giữa các mô-đun. Trong trường hợp tốt nhất, sự giao tiếp chỉ là các phần tử dữ liệu gửi đi và các cờ điều khiển;

4- Chọn kích cỡ hợp lý: các mô-đun hạn chế từ 50 đến 100 dòng lệnh;

5- Bảo đảm kết dính: mở rộng một thành phần đến mức có thể sao cho mọi lệnh trong một mô-đun thuộc cùng một chức năng. Ta có thể mô tả chính xác chức năng mà mô-đun thực hiện với số từ ít nhất, với tối thiểu số lượng các từ "và", "hoặc";

6- Tạo các mô-đun sử dụng chung: thiết kế các mô-đun con mức thấp sao cho có thể được nhiều mô-đun con khác gọi đến. Ý tưởng này phục vụ yêu cầu kích thước hợp lý nhằm làm cho mô-đun đủ nhỏ để các nhà lập trình có thể hiểu được nó, có thể dễ sửa đổi nó và đủ nhỏ để trở thành những thao tác cơ bản mà nhiều thành phần khác có thể sử dụng lại được. Sự ghép nối có thể được ghi ra trong biểu đồ cấu trúc thông qua việc di chuyển các ghép nối dữ liệu hay các cờ giữa các mô-đun.

TÓM TẮT CHƯƠNG

Kiến trúc phần mềm được hiểu như một cấu trúc tổng thể của phần mềm và qua đó cung cấp một sự tích hợp về mặt khái niệm của một hệ thống. Ở

dạng đơn giản nhất, kiến trúc phần mềm là cấu trúc phân cấp của các thành phần chương trình qua đó thể hiện sự tương tác giữa chúng với nhau và các cấu trúc dữ liệu mà chúng sử dụng.

Kiến trúc có một vai trò quan trọng trong phát triển phần mềm. Nó là công cụ giao tiếp giữa những người có liên quan đến phát triển hệ thống; dùng để phân tích hệ thống và giúp sử dụng lại ở quy mô lớn. Mục tiêu chủ yếu của thiết kế kiến trúc là phát triển một cấu trúc chương trình mô-đun và biểu diễn mối quan hệ điều khiển giữa các mô-đun.

Thiết kế kiến trúc là giai đoạn đầu của quá trình thiết kế. Nó biểu diễn sự kết nối giữa đặc tả yêu cầu và các tiến trình thiết kế. Nó thường tiến hành song song với các hoạt động đặc tả phần mềm và bao gồm việc xác định các thành phần hệ thống và giao tiếp giữa chúng.

Thiết kế kiến trúc tập trung vào việc biểu diễn cấu trúc của các thành phần phần mềm, các thuộc tính của chúng và các tương tác giữa chúng.

Kiến trúc phần mềm được mô tả ở dạng các mô hình khác nhau. Các mô hình kiến trúc được phát triển thường có thể nhìn nhận dưới một số mặt như kiến trúc tĩnh, tiến trình động, các giao diện đưa ra các dịch vụ mà mỗi hệ con cung cấp, mối liên kết giữa các hệ con hay giữa các thành phần.

Có hai phương pháp chính để tạo kiến trúc là phân hoạch kiến trúc theo kiểu ngang hay kiểu dọc và tạo kiến trúc từ sơ đồ luồng dữ liệu. Chú ý rằng, cả hai kiểu phân hoạch đó đều dễ kiểm thử, bảo trì, dễ mở rộng nhưng có hạn chế là có thể có hiệu ứng phụ khi sửa đổi.

Ba hoạt động chung nhất cho mọi tiến trình thiết kế kiến trúc là:

1-Cấu trúc hệ thống thành một số hệ thống con;

2-Thiết lập một mô hình chung mô tả mối quan hệ điều khiển giữa các phần trong hệ thống;

3-Phân rã các mô-đun.

Có nhiều cách phân tích đặc tả yêu cầu phần mềm thành đặc tả kiến trúc. Mặc dầu mỗi cách đều có cách tiếp cận khác nhau để suy ra kiến trúc, nhưng tất cả đều thừa nhận tầm quan trọng của cái nhìn chung về phần mềm. Phương

pháp phổ cập để tạo kiến trúc là phương pháp tạo kiến trúc dựa vào sơ đồ luồng dữ liệu.

Thiết kế hướng luồng dữ liệu thường được gọi là "thiết kế có cấu trúc" có nguồn gốc từ những khái niệm ban đầu nhấn mạnh vào tinh mô-đun, với cách tiếp cận top_down và lập trình có cấu trúc. Phương pháp thiết kế này đã và đang được áp dụng cho một phạm vi rộng các miền ứng dụng.

Muốn bảo đảm chất lượng thiết kế kiến trúc phần mềm, cần phải tuân theo một số tiêu chuẩn và nguyên tắc thiết kế nhất định.

CÂU HỎI CHƯƠNG 5

1- Kiến trúc có phải thành phần hoạt động trong hệ thống? Tại sao người ta phải quan tâm đến đặc tả thiết kế kiến trúc phần mềm?

2. Kiến trúc phần mềm được thể hiện qua hai đặc trưng quan trọng nào của chương trình máy tính?

3.Ý nghĩa của cấu trúc dữ liệu?

4. Mục tiêu chủ yếu của thiết kế kiến trúc là gì?

5. Thiết kế kiến trúc tập trung vào những công việc gì?

6. Trong công nghệ xây dựng kiến trúc hệ thống, ở Bước1- Phân rã hệ thống thành tập các hệ thống con, đối với hệ thống lớn người ta thường dùng cách tiếp cận theo hướng đối tượng hay hướng cấu trúc?, tại sao?

7. Trong quy trình thiết kế kiến trúc chương trình, bước đầu tiên có thể chỉ cần xây dựng mô hình luồng dữ liệu nghiệp vụ có được không?, tại sao?

8. Biểu đồ chung nhất để biểu diễn mô hình kiến trúc của một hệ thống là gì? mô tả sơ lược về nó?

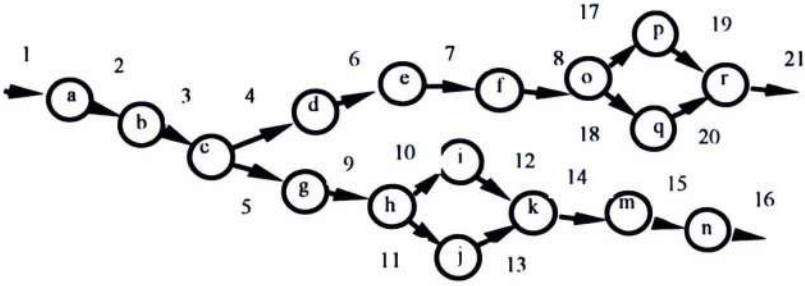
9. Có những phương pháp nào để tạo kiến trúc phần mềm, nêu rõ các điều kiện để sử dụng mỗi phương pháp đó?

10. Nêu một số tiêu chuẩn chính cho thiết kế kiến trúc phần mềm? Dựa vào đó Anh/chị hãy đưa ra một số hướng dẫn thiết kế kiến trúc của riêng mình?

BÀI TẬP CHƯƠNG 5

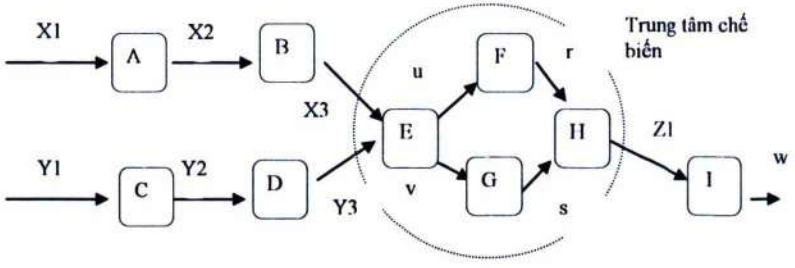
Bài tập 5.1

Đặc tả kiến trúc phần mềm từ sơ đồ luồng dữ liệu sau:



Bài tập 5.2

Đặc tả kiến trúc phần mềm từ sơ đồ luồng dữ liệu sau:



Chương 6

ĐẶC TẢ KIẾN TRÚC MỘT SỐ HỆ THỐNG ĐƯỢC THIẾT KẾ THEO HƯỚNG HIỆN ĐẠI

Để đáp ứng yêu cầu thiết kế cho những hệ thống hiện đại, người ta đã mất nhiều công sức nghiên cứu kiến trúc phù hợp với chúng. Kết quả, đặc tả kiến trúc của một số hệ thống được thiết kế theo hướng hiện đại như hướng phân tán, đối tượng và hướng dịch vụ đã thể hiện những nét riêng biệt, độc đáo, mang tính công nghệ cao.

Chương này trình bày phần đặc tả kiến trúc một số hệ thống được thiết kế theo hướng hiện đại.

Học viên cần nắm vững kỹ năng đặc tả kiến trúc hai hệ thống phổ biến được thiết kế theo hướng hiện đại, đó là hướng phân tán, hướng đối tượng và hướng dịch vụ.

6.1. Đặc tả kiến trúc hệ thống được thiết kế theo hướng phân tán

6.1.1. Sơ lược về hệ thống phân tán

Việc quản lý các luồng dữ liệu trên cơ sở *mạng cục bộ (Local Arear Network-LAN)* với kiến trúc máy quản lý file đơn giản đã thể hiện chưa đáp ứng được yêu cầu thực tế phức tạp, mặc dù khi kết nối các máy tính thành mạng, NSD đã có thể dùng chung dữ liệu và các tài nguyên khác của mạng như máy in, máy fax,... sẽ rẻ hơn khi dùng riêng lẻ. Vì thế, các hệ thống phân tán được thiết kế và phát triển. Chúng có một ý nghĩa thực tiễn to lớn đối với các hoạt động hàng ngày của tổ chức và là một thành tựu mới của việc phát triển một HTTT.

Thiết kế các hệ thống phân tán có nhiều điểm giống với các thiết kế các hệ thống tại một vị trí. Sự khác nhau đầu tiên đó là hệ thống phân tán được phân bố ở một số địa điểm khác nhau. Khi thiết kế hệ thống phân tán, người ta

quan tâm đến nhiều mặt như tính khả thi, tính sẵn sàng, sự sống của hệ thống khi nó được triển khai ở nhiều địa điểm. Các hệ thống phân tán gồm nhiều máy trạm/khách, máy chủ/dịch vụ, mạng, nhiều địa điểm, nhiều dữ liệu,... Vì ở nhiều địa điểm có nguy cơ xảy ra sự cố nên cần phải sử dụng nhiều chiến lược khác nhau để thiết kế và triển khai chúng.

Khi thiết kế hệ thống phân tán cần chú ý đến tính cân đối giữa ba yếu tố có thể tác động đến nó là *tính khả thi, sự sống và tính sẵn sàng* của hệ thống. Để có được thiết kế hiệu quả, cần nắm vững những đặc trưng của kiến trúc được sử dụng để trợ giúp các hệ thống phân tán và chia sẻ dữ liệu. Nói khác đi, cần quyết định xem có cần phân tán dữ liệu và các xử lý ở một số địa điểm (trạm) không và nếu chọn phương án phân tán thì cần phải lưu ý những điểm sau:

1- Xác định kiến trúc mô hình phân tán tổng thể bao gồm định vị các trạm cần phân tán, loại hình phân tán sử dụng cho mỗi trạm;

2- Tiến hành cân đối các yếu tố được phân tán bao gồm các phần tử dữ liệu và các hoạt động xử lý trên mỗi trạm;

3- Thiết kế CSDL phân tán;

4- Thiết kế các hệ thống chương trình tương ứng.

Khi thiết kế hệ thống phân tán cũng cần luôn ghi tâm những ưu thế và hạn chế của hệ thống phân tán:

a-Về ưu điểm:

1- Tăng cường khả năng của hệ thống liên quan đến tính dư thừa;

2- Kiểm soát dữ liệu địa phương theo hướng hoàn thiện sự tích hợp và quản trị dữ liệu từ xa;

3- Tăng cường các thành phần ứng dụng và CSDL mà không làm cản trở NSD;

4- Đáp ứng nhanh hầu hết các ứng dụng sử dụng dữ liệu ở mỗi trạm.

b-Về hạn chế:

1- Phần mềm đắt và phức tạp;

2- Phải xử lý mọi thay đổi thông báo trong mọi địa điểm;

3- Khó kiểm soát được tính toàn vẹn dữ liệu với nhiều bản sao dữ liệu và được phân bố rộng rãi;

4- Đáp ứng chậm nhu cầu trong trường hợp các phần mềm ứng dụng không được phân bố phù hợp với việc sử dụng chúng.

Để đáp ứng được mục tiêu cung cấp dịch vụ truy nhập dữ liệu cho NSD ở mọi nơi khác nhau, hệ thống phân tán cần có tính "*trong suốt địa phương*". Tính trong suốt địa phương được hiểu là một NSD ở bất kỳ một vị trí (trạm) nào trên mạng khi yêu cầu dữ liệu đều không cần biết các dữ liệu họ cần được lưu trữ ở đâu. Các yêu cầu lấy dữ liệu hay cập nhật dữ liệu ở một trạm nào đó đều được đáp ứng tự động bằng cách hệ thống gửi các thông tin cần thiết đến trạm này.

6.1.2. Tổ chức hệ thống theo mạng địa phương

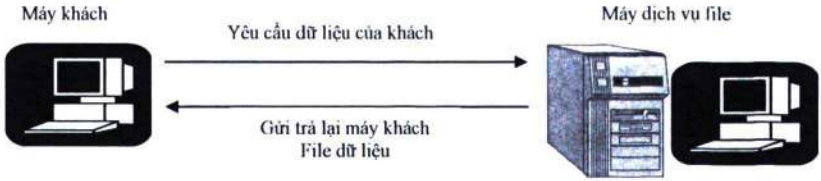
Bằng việc kết nối bên trong giữa các máy tính, người dùng có thể sử dụng chung dữ liệu và các tài nguyên khác của mạng như máy in, máy fax,... như vậy sẽ rẻ hơn khi dùng riêng lẻ.

Mạng LAN sẽ trợ giúp một mạng máy tính cá nhân với các kho dữ liệu riêng của nó và có thể chia sẻ các thiết bị và phần mềm trên nó. Một máy tính được gán nhiệm vụ của một máy chủ để lưu trữ CSDL và các ứng dụng. Các thành phần của hệ quản trị CSDL sẽ trợ giúp việc truy nhập từ nhiều người dùng vào CSDL dùng chung

Trong môi trường LAN, tất cả thao tác dữ liệu đều diễn ra ở máy trạm, ở đó dữ liệu được yêu cầu. Một hay một số máy dịch vụ file được gắn vào mạng LAN. Một máy dịch vụ file là một thiết bị quản lý các hoạt động file và phục vụ các máy trạm được kết nối trong mạng LAN.

Trong cấu hình của máy dịch vụ file, mỗi máy dịch vụ file có một phần đĩa cứng dành cho mỗi máy trạm. Chương trình trên máy trạm có thể tham chiếu đến các file trên đĩa này bằng một đặc tả đường dẫn đến và mọi thư mục cùng file trên nó [17].

Khi sử dụng một CSDL trong môi trường máy dịch vụ file, mỗi máy trạm được phép sử dụng chương trình ứng dụng CSDL trên nó. Như vậy, có một CSDL trên máy dịch vụ file và nhiều bản sao của nó hoạt động bình thường trên mỗi máy trạm đang hoạt động.



Hình 6.1. Kiến trúc máy dịch vụ file

Đặc trưng nguyên thủy của mạng LAN dựa trên máy trạm là tất cả mọi thao tác dữ liệu được thực hiện trên máy trạm, không phải trên máy dịch vụ file. Máy dịch vụ file đơn giản như một thiết bị lưu trữ dữ liệu dùng chung và là sự mở rộng của máy trạm. Như vậy, khi các máy trạm làm việc và có yêu cầu, máy dịch vụ sẽ gửi toàn bộ file tương ứng qua mạng đến máy trạm, và ở đó các thao tác dữ liệu được thực hiện. Các hoạt động an toàn cũng được thực hiện tại máy trạm.

Chú ý rằng, kiến trúc máy dịch vụ file trên mạng LAN có những hạn chế sau:

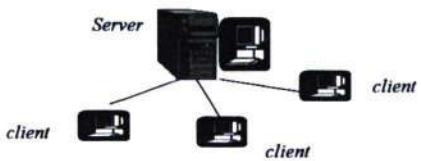
1- Sự di chuyển dữ liệu sẽ diễn ra quá nhiều trên mạng;

2- Các máy trạm cần có cấu hình phải đủ mạnh;

3- Việc kiểm soát dữ liệu là phi tập trung.

6.1.3. Tổ chức hệ thống theo kiến trúc Client-Server

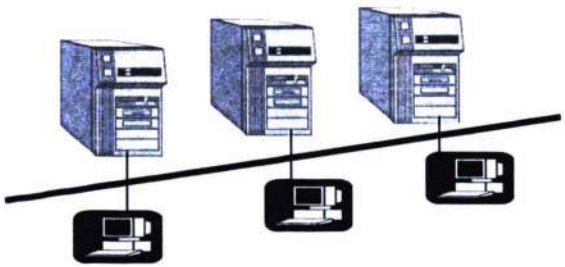
Thuật ngữ Client-Server chỉ khái niệm chia sẻ công việc liên quan trong xử lý dữ liệu giữa máy khách (Client) và máy phục vụ (Server), trong đó máy phục vụ có công suất mạnh hơn.



Hình 6.2. Kiến trúc Client-Server với cấu hình đơn máy phục vụ

Kiến trúc Client-Server là một cải tiến trong hệ thống dựa trên mạng LAN. Kiến trúc này hướng tới các dữ liệu và xử lý ứng dụng được phân chia

giữa máy khách (máy trạm) và máy dịch vụ. Máy khách thường quản lý các giao diện và trình diễn dữ liệu, còn máy dịch vụ CSDL đại diện cho việc lưu trữ CSDL và truy nhập đến nó, xử lý các truy vấn. Kiến trúc Client-Server tiêu biểu được đặc tả trên hình 6.3, trong đó máy khách gửi yêu cầu dữ liệu của mình tới máy dịch vụ và máy dịch vụ trả lại máy khách kết quả yêu cầu.

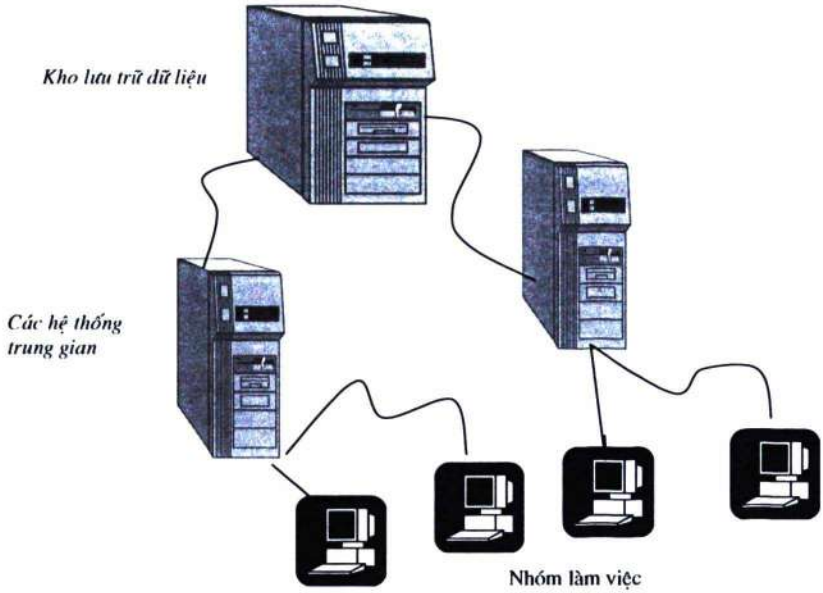


Hình 6.3. Kiến trúc Client-Server với cấu hình đa máy phục vụ

Trong kiến trúc Client-Server, tất cả các hoạt động phục hồi, an toàn CSDL và quản lý truy nhập tương tranh đều tập trung ở máy dịch vụ. Các chức năng CSDL trung tâm thường được gọi là "máy CSDL" trong một môi trường Client-Server.

Ở máy dịch vụ, mọi yêu cầu về dữ liệu được thực hiện, và chỉ những dữ liệu kết quả đáp ứng các yêu cầu mới được gửi về máy khách qua mạng. Như vậy, máy dịch vụ cung cấp mọi dịch vụ CSDL chung cho các máy khách.

Ở đây, kho lưu trữ dữ liệu làm nhẹ bớt gánh nặng dữ liệu của những máy phục vụ khác. Máy phục vụ chuyên dụng là máy chỉ hoạt động như một máy phục vụ chứ không kiêm luôn vai trò của máy khách hay trạm làm việc. Máy phục vụ chuyên dụng có tính chuyên dụng vì chúng được tối ưu hoá để phục vụ nhanh những yêu cầu của khách hàng trên mạng, cũng như để bảo đảm an toàn cho cả file và thư mục. Mạng dựa trên máy phục vụ đã trở thành mô hình chuẩn cho hoạt động mạng. Vì mạng phát triển nhanh cả quy mô lẫn lưu lượng nên cần nhiều máy phục vụ thay vì chỉ có một. Phân phối tác vụ giữa nhiều máy phục vụ sẽ bảo đảm mỗi tác vụ sẽ được thi hành theo cách thức hiệu quả nhất có thể có [8].



Hình 6.4. Kiến trúc Client-Server với cấu hình có máy phục vụ chuyên dụng

Các ứng dụng xây dựng trên cơ sở kiến trúc Client-Server cũng khác với hệ thống CSDL tập trung trên một máy lớn. Điều khác cơ bản đó là, mỗi máy khách là một phần thông minh của hệ thống xử lý ứng dụng. Nói cách khác, chương trình ứng dụng được người dùng thực hiện trên máy khách mà không phải trên máy dịch vụ. Còn máy dịch vụ quản lý tất cả hoạt động truy nhập dữ liệu và các chức năng kiểm tra. Trong khi đó, trong môi trường máy lớn, tất cả các bộ phận của HTTT được thực hiện và quản lý trên máy lớn.

Một ưu điểm khác của kiến trúc Client-Server là khả năng ghép nối môi trường máy khách với môi trường máy dịch vụ. Máy khách có thể gồm nhiều loại khác nhau. Điều đó có nghĩa là, nó có thể dùng một hệ thống ứng dụng bất kỳ để sinh ra lệnh gửi yêu cầu dữ liệu đến đến máy dịch vụ (chương trình có thể viết bằng các ngôn ngữ khác nhau) miễn là có giao diện chương trình ứng dụng cho máy CSDL.

Tóm lại kiến trúc Client-Server có các ưu điểm sau:

1- Cho phép nhận được nhiều lợi ích từ công nghệ máy trạm mini;

2- Cho phép thực hiện được hầu hết các xử lý gần nguồn dữ liệu được xử lý, nhờ vậy rút ngắn được thời gian và chi phí lưu thông trên mạng;

3- Tạo điều kiện sử dụng các giao diện đồ họa và kỹ thuật trình diễn trực quan thường sẵn có đối với các máy trạm;

4- Khuyến khích chấp nhận các hệ thống mở.

6.1.4. Các đặc trưng của máy dịch vụ file và kiến trúc Client-Server

Cả hai mô hình máy dịch vụ file và cấu trúc Client-Server đều sử dụng máy tính cá nhân, máy trạm và nối với nhau bằng mạng LAN. Trong khi kiến trúc dịch vụ file trợ giúp phân tán dữ liệu thì kiến trúc Client-Server trợ giúp cả phân tán dữ liệu và phân tán xử lý.

Bảng 6.1. Sự khác nhau giữa máy dịch vụ file và kiến trúc Client-Server

Đặc trưng	**Máy dịch vụ file**	**Kiến trúc Client-Server**
Xử lý	Chỉ ở máy khách	Cả máy khách và máy dịch vụ
Truy nhập dữ liệu đồng thời	Thấp, mỗi máy khách thực hiện	Cao, máy dịch vụ đảm nhiệm
An toàn và toàn vẹn CSDL	Thấp, máy khách quản lý	Cao, máy dịch vụ đảm nhiệm
Sử dụng mạng	File lớn, chuyển cả file	Truyền dữ liệu nhiều mức
Bảo trì phần mềm	Thấp, chỉ ở máy dịch vụ	Hỗn hợp, một số phần mềm có thể gửi đến máy khách
Phần cứng và hệ thống, sự mềm dẻo phần mềm	Ghép nối máy khách và máy dịch vụ và có thể phối hợp	Ghép nối máy khách và máy dịch vụ và có thể phối hợp

6.1.5. Các thành phần của mô hình Client-Server

Hiện nay, người ta đã đưa vào mô hình Client-Server nhiều chức năng hệ thống ứng dụng khác nhau dựa trên ba thành phần sau đây [17]:

1- *Quản trị dữ liệu*: các chức năng này quản lý mọi tương tác giữa phần mềm, file và CSDL, bao gồm việc lấy dữ liệu, truy vấn, cập nhật, an toàn, kiểm tra tương tranh và phục hồi.

2- *Trình diễn dữ liệu*: chức năng này quản lý giao diện giữa phần mềm, người dùng và hệ thống, bao gồm hiển thị, in các biểu báo và thẩm định đầu vào hệ thống.

3- *Xử lý dữ liệu*: chức năng này chuyển đổi cái vào thành cái ra bao gồm từ tổng hợp đơn giản đến các mô hình toán học phức tạp.

Các kiến trúc Client-Server khác nhau phân tán các chức năng kể trên cho từng máy khách, máy dịch vụ hay cả hai. Theo cách phân tán này có thể có đến 27 mô hình khác nhau, trong đó có 6 mô hình là phổ dụng hơn cả (các bảng từ 6.2 đến 6.7).

Công nghệ hiện tại cho phép phát triển ứng dụng sử dụng khi sử dụng một trong số các mô hình trên thông qua công cụ CASE mà không cần phải tạo ra mã chương trình riêng cho mỗi loại.

6.1.6. Những dạng tiên tiến phổ dụng của kiến trúc Client-Server

6.1.6.1. Trình diễn thông tin phân tán

Hình thức trình diễn phân tán của kiến trúc Client-Server được sử dụng để làm mới các ứng dụng trên máy dịch vụ và được gửi cho máy khách. Trong kiến trúc Client-Server, công việc gọi "máy quét màn hình" làm việc trên máy khách để định dạng lại một cách đơn giản các dữ liệu do máy dịch vụ quản lý. Kết quả này làm dễ dàng việc sử dụng báo cáo, biểu mẫu và giao diện mà không phá hủy hoặc phải viết lại hệ thống cũ. Trình diễn phân tán đã hạn chế được sự hoạt động của các biểu mẫu, báo cáo đang tồn tại, và khi cần những đơn thể trình diễn trên cả máy khách và máy dịch vụ có thể thay đổi và bảo trì đồng thời.

Bảng 6.2. Trình diễn thông tin phân tán

Chức năng	Máy khách	Máy dịch vụ
Quản lý dữ liệu		Quản lý mọi dữ liệu
Phân tích dữ liệu		Phân tích mọi dữ liệu
Trình diễn dữ liệu	Dữ liệu trình diễn trên máy dịch vụ được định dạng để trình diễn cho người dùng	Dùng dữ liệu gửi cho khách, sử dụng công nghệ trình diễn của máy dịch vụ

6.1.6.2. Trình diễn từ xa

Kiểu trình diễn từ xa của mô hình Client-Server đặt tất cả các chức năng trình diễn dữ liệu trên máy khách nên phần mềm trên máy khách có mọi khả năng trình diễn những dữ liệu định dạng. Kiến trúc tạo ra một sự mềm dẻo rất lớn so với kiến trúc trình diễn phân tán. Khi người dùng cần thay đổi các biểu mẫu, báo cáo hay nội dung mới thì chỉ cần bảo trì trên máy khách.

Ngày nay, định dạng siêu văn bản (HTMT) trở thành định dạng chung cho hệ mạng toàn cầu Internet. Vì vậy một máy khách trong hệ này được trang bị sẵn các phần mềm trình duyệt HTML (như Netscaps, Internet Explorer,...) có thể trình duyệt mọi thông tin lấy về từ mọi máy dịch vụ trên hệ thống mà nó có thể kết nối được.

Bảng 6.3. Trình diễn từ xa

Chức năng	Máy khách	Máy dịch vụ
Quản lý dữ liệu		Quản lý mọi dữ liệu
Phân tích dữ liệu		Phân tích mọi dữ liệu
Trình diễn dữ liệu	Dữ liệu phân tích trên máy dịch vụ được định dạng để trình diễn cho người dùng	

6.1.6.3. Quản lý dữ liệu từ xa

Hình thức quản lý dữ liệu từ xa của kiến trúc Client-Server đặt mọi phần mềm ở máy khách, trừ các phần mềm quản lý dữ liệu. Trên một hệ thống mạng tốc độ cao, tất cả các dữ liệu cần cho sự phân tích (mà không phải cho sự trình diễn) đều có thể truyền từ máy dịch vụ sang máy khách. Trên máy khách có thể sử dụng các phần mềm bất kỳ (Excel, Matlab,...) để xử lý dữ liệu có được. Như vậy CSDL trên máy dịch vụ được sử dụng chung nhưng nó vẫn quản lý tập trung. Ngày nay, do sự phát triển của kỹ thuật phần cứng, các máy khách là đủ mạnh (cả phần cứng và phần mềm), đủ khả năng để lưu trữ được các dữ liệu lớn và tiến hành xử lý cần thiết.

Bảng 6.4. Quản lý dữ liệu từ xa

Chức năng	**Máy khách**	**Máy dịch vụ**
Quản lý dữ liệu		Quản lý mọi dữ liệu
Phân tích dữ liệu	Dữ liệu thô được lấy từ máy dịch vụ và được phân tích	
Trình diễn dữ liệu	Trình diễn tất cả dữ liệu	

6.1.6.4. Phân tán chức năng

Kiến trúc Client-Server với chức năng phân tán các chức năng phân tích tên cả máy khách và máy dịch vụ, để toàn bộ phần trình diễn dữ liệu trên máy khách, tất cả chức năng quản lý dữ liệu trên máy chủ. Kiến trúc này cho phép cài đặt các chức năng phân tích trên các máy mà có chi phí hiệu quả nhất. Chẳng hạn, những phân tích đòi hỏi nhiều dữ liệu có thể đặt trên máy dịch vụ mà trên đó lưu trữ phần lớn các dữ liệu cần thiết cho việc phân tích sẽ giảm lưu lượng thông tin cần truyền trên mạng.

Tuy nhiên, việc phát triển, kiểm thử và bảo trì không tránh khỏi khó khăn vì nó liên quan đến việc bảo đảm sự phối hợp nhất quán giữa các chức năng phân tích được phân tán cả trên máy khách và trên máy dịch vụ.

Bảng 6.5.Phân tán chức năng

Chức năng	Máy khách	Máy dịch vụ
Quản lý dữ liệu		Quản lý mọi dữ liệu
Phân tích dữ liệu	Dữ liệu được lấy và phân tích từ máy dịch vụ	Dữ liệu được lấy và phân tích từ máy dịch vụ và sau đó truyền cho máy khách
Trình diễn dữ liệu	Tất cả dữ liệu được phân tích cả trên máy dịch vụ và máy khách	

6.1.6.5. Cơ sở dữ liệu phân tán (distributed database)

Kiến trúc Client-Server với CSDL phân tán đặt mọi chức năng trên máy khách, trừ phần dữ liệu và chức năng quản lý được dự kiến cho máy dịch vụ. Trong trường hợp này, mỗi máy khách (máy trạm) được lưu trữ các dữ liệu cần thiết thường xuyên cho các xử lý của nó. Chỉ dữ liệu dùng chung mới đặt trên máy dịch vụ với cùng một truy vấn. Khái niệm về phân đoạn dữ liệu theo chiều dọc hay phân đoạn dữ liệu theo chiều ngang được sử dụng để quyết định xem dữ liệu nào được lưu trữ trên máy khách hay máy dịch vụ nào.

Nhiều công cụ đã được phát triển (như Designer 2000 của Oracle) để làm đơn giản việc thiết kế và triển khai các hệ thống CSDL phân tán.

Bảng 6.6. Phân tán dữ liệu

Chức năng	Máy khách	Máy dịch vụ
Quản lý dữ liệu	Quản lý dữ liệu địa phương	Chia sẻ quản lý dữ liệu trên máy dịch vụ
Phân tích dữ liệu	Dữ liệu được lấy từ cả máy dịch vụ và máy khách để phân tích	
Trình diễn dữ liệu	Tất cả dữ liệu	

6.1.6.6. Xử lý phân tán (distributed procesing)

Kiến trúc Client-Server với xử lý phân tán phối hợp các đặc trưng tốt nhất của chức năng phân tán và CSDL phân tán bằng cách liên kết chúng lại trên cả máy khách và máy dịch vụ, và chỉ để lại chức năng trình dữ liệu trên máy khách. Mô hình này cho phép định vị một cách mềm dẻo cả chức năng phân tích và dữ liệu ở nơi mà chúng hoạt động là tốt nhất. Tuy nhiên, cũng như các mô hình khác, mô hình này không tránh khỏi một số khó khăn như nó vốn có từ các mô hình đã nêu trên.

Bảng 6.7. Xử lý phân tán

Chức năng	**Máy khách**	**Máy dịch vụ**
Quản lý dữ liệu	Quản lý dữ liệu địa phương	Chia sẻ quản lý dữ liệu trên máy dịch vụ
Phân tích dữ liệu	Dữ liệu được lấy từ cả máy dịch vụ và máy khách để phân tích	Dữ liệu được lấy và phân tích từ máy dịch vụ và sau đó truyền cho máy khách để phân tích tiếp và trình diễn
Trình diễn dữ liệu	Tất cả dữ liệu	

Những mô hình kiến trúc nêu trên cho các nhà thiết kế một phạm vi rộng rãi để lựa chọn một mô hình thích hợp đối với mỗi trường hợp cụ thể. Cũng giống như các thiết kế vật lý khác, những chuẩn mực của tổ chức và các ràng buộc mà tổ chức đặt ra (thời gian đáp ứng, trình độ NSD,...) phải được xem xét khi lựa chọn một mô hình thích hợp.

6.1.7. Sự cân bằng giữa các yếu tố trong hệ phân tán

Đối tượng chủ yếu của CSDL phân tán là cung cấp dịch vụ truy nhập dữ liệu cho NSD ở mọi nơi khác nhau. Để đáp ứng được mục tiêu đó, hệ thống phân tán phải có được "*tính trong suốt địa phương*". Tính trong suốt địa phương được hiểu là một NSD ở bất kỳ một địa phương nào khi yêu cầu dữ liệu thì họ không cần biết các dữ liệu họ cần được lưu trữ ở đâu.

Mỗi yêu cầu bất kỳ để lấy dữ liệu hay cập nhật dữ liệu ở một trạm nào đó đều được đáp ứng tự động bằng hệ thống gửi các thông tin cần thiết đến trạm này.

6.2. Đặc tả kiến trúc hệ thống được thiết kế theo hướng đối tượng

6.2.1. Mô hình hóa kiến trúc hệ thống

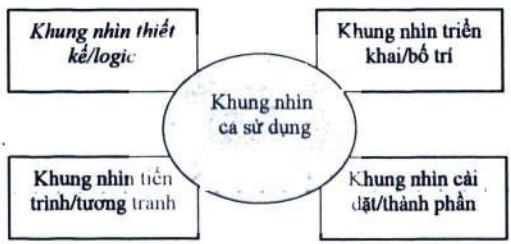
Trong chương 5, chúng ta đã biết rằng kiến trúc hệ thống là cấu trúc tổ chức của hệ thống. Kiến trúc gồm nhiều bộ phận có thể ở nhiều mức khác nhau, tương tác với nhau thông qua các giao diện, các mối quan hệ kết nối và các ràng buộc để kết hợp chúng thành một thể thống nhất [11].

Kiến trúc hệ thống hướng đối tượng được chia thành hai loại:

1- Kiến trúc logic: *chỉ ra các lớp và đối tượng, các quan hệ và sự cộng tác để hình thành chức năng của hệ thống. Kiến trúc logic được mô tả bởi các sơ đồ ca sử dụng, sơ đồ lớp và các sơ đồ tương tác.*

2- Kiến trúc vật lý: *đề cập đến việc mô tả chi tiết hệ thống về phương diện phần cứng và phần mềm của hệ thống. Đồng thời nó cũng mô tả cấu trúc vật lý và sự phụ thuộc của các mô-đun cộng tác trong cài đặt những khái niệm đã được định nghĩa trong kiến trúc logic.*

Trong UML, kiến trúc của hệ thống phần mềm chuyên sâu (cho ta một cách nhìn khái quát nhất về hệ thống phần mềm ở các góc độ khác nhau) được mô tả theo 5 loại khung nhìn (views) khác nhau [11]. Mỗi khung nhìn phản ánh về một khía cạnh của tổ chức và cấu trúc của hệ thống, tập trung vào từng mặt cụ thể giúp cho ta hiểu và sử dụng hệ thống tốt nhất. Mỗi khung nhìn thường được thể hiện trong một số sơ đồ nhất định.



Hình 6.5. Mô hình hoá kiến trúc hệ thống

1- Khung nhìn ca sử dụng (Use case view)*: chứa các tác nhân, ca sử dụng, sơ đồ ca sử dụng (khía cạnh tĩnh của khung nhìn) trong hệ thống. Chúng cũng có thể bao gồm vài sơ đồ trình tự, sơ đồ cộng tác (khía cạnh động của khung nhìn) và gói. Khung nhìn ca sử dụng tập trung vào mức cao của cái hệ thống sẽ làm, không quan tâm đến hệ thống làm như thế nào.*

2- Khung nhìn thiết kế (design view) *tập trung vào hệ thống cài đặt hành vi trong ca sử dụng như thế nào. Nó bao gồm các lớp, sơ đồ lớp, sơ đồ đối tượng (khía cạnh tĩnh của khung nhìn), sơ đồ tương tác, sơ đồ hoạt động, sơ đồ biến đổi trạng thái (khía cạnh động của khung nhìn) và các gói. Trong khung nhìn, người ta quan tâm đến hai lớp quan trọng là lớp phân tích (lớp biên, lớp điều khiển, lớp dữ liệu) và lớp thiết kế (lớp phụ thuộc vào ngôn ngữ). Khung nhìn này tập trung vào cấu trúc logic của hệ thống nên còn được gọi là* khung nhìn logic (logical view). *Khung nhìn này tập trung vào cấu trúc của hệ thống. Nhờ nó, người ta nhận ra các bộ phận cơ bản cấu thành hệ thống thể hiện mọi quá trình trao đổi, xử lý thông tin cơ bản trong hệ thống.*

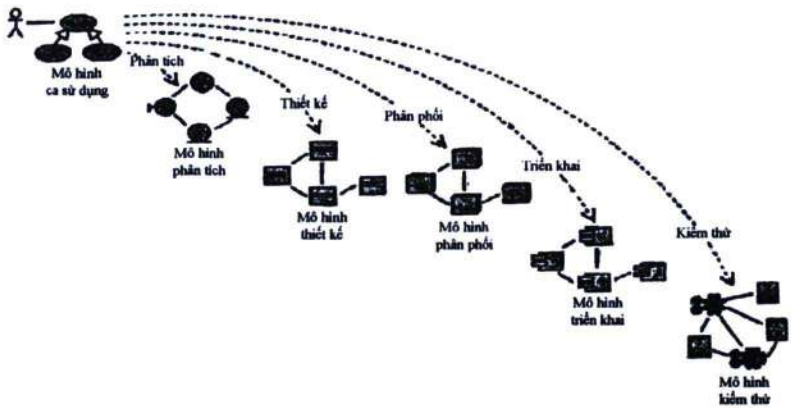
3- Khung nhìn tiến trình (process view) *biểu diễn sự phân chia các luồng thực hiện công việc, các lớp đối tượng cho các* tiến trình *và sự đồng bộ giữa các* luồng *(thread) trong hệ thống. Khung nhìn này tập trung vào các nhiệm vụ tương tranh, tương tác với nhau trong hệ thống đa nhiệm. Nó chủ yếu diễn đạt hiệu năng, quy mô và năng lực thông qua của hệ thống.*

4- Khung nhìn thành phần/cài đặt (component/implementation view) *bao gồm các thành phần (là mô-đun vật lý hay các file) để lắp ráp thành hệ thống vật lý. Khung nhìn này hướng đến việc quản lý cấu hình của hệ thống. Khung nhìn này bao gồm thành phần, sơ đồ thành phần và gói.*

5- Khung nhìn bố trí/triển khai (Deployment view) *bao gồm các nút tạo nên kết cấu phần cứng mà trên đó hệ thống vận hành. Khung nhìn này chủ yếu hướng đến sự phân tán và cài đặt cụ thể của hệ thống, tức là liên quan đến triển khai vật lý của hệ thống. Khung nhìn triển khai chỉ ra các tiến trình và thiết bị trên mạng và các kết nối vật lý giữa chúng. Sơ đồ triển khai cũng hiển thị tiến trình và chỉ ra tiến trình nào chạy trên máy nào. Khung nhìn này được thể hiện trong các sơ đồ triển khai/bố trí các nút của hệ thống.*

Quá trình phát triển phần mềm hướng đối tượng có thể sử dụng các công cụ khác nhau. Quá trình phát triển phần mềm có 3 đặc trưng cơ bản sau:

- Ca sử dụng điều khiển quá trình phát triển;
- Lấy kiến trúc làm trung tâm;
- Tiến trình phát triển là tiến trình lặp và tăng dần.



Hình 6.6. Ca sử dụng điều khiển quá trình phát triển phần mềm

Một đặc trưng nổi bật của quá trình phát triển là lấy kiến trúc làm trung tâm [11].

Vai trò của kiến trúc hệ thống phần mềm giống một khung nền dựa trên đó phần mềm được xây dựng và phát triển đến hoàn thiện. Khái niệm kiến trúc phần mềm chứa đựng những khía cạnh tĩnh và động có ý nghĩa nhất định đối với hệ thống. Nó được phát triển dựa theo yêu cầu của tổ chức, theo cảm nhận của người dùng và các tổ chức có liên quan khác. Mặt khác, nó cũng chịu ảnh hưởng của rất nhiều nhân tố khác, chẳng hạn như phần mềm của hệ thống, các khối xây dựng dùng lại được có sẵn, các cân nhắc về điều kiện triển khai, các hệ thống có sẵn trong môi trường tương tác với nó, và cả các yêu cầu phi chức năng. Kiến trúc là một cái nhìn thiết kế tổng thể những đặc điểm quan trọng nhất về hệ thống phần mềm khi tạm bỏ qua các chi tiết.

Mọi sản phẩm phần mềm đều bao gồm chức năng và hình thức thể hiện. Hai yếu tố này phải cân bằng với nhau để đem lại kết quả tốt nhất. Chức năng

tương ứng với ca sử dụng và hình thức thể hiện tương ứng với kiến trúc. Do đó, việc lựa chọn các ca sử dụng để phát triển được định hướng theo kiến trúc và phải phù hợp với kiến trúc. Nói cách khác, kiến trúc phải cung cấp chỗ dựa cho việc thực hiện các ca sử dụng ngay khi bắt đầu tiến trình phát triển hệ thống và cả trong tương lai. Để có được bản mẫu cho kiến trúc, người phân tích phải có hiểu biết chung về các chức năng chính, đó là các ca sử dụng chính yếu. Chúng là các ca sử dụng mang ý nghĩa nhất, tạo nên các chức năng chủ yếu của hệ thống và thường ít thay đổi trong quá trình phát triển.

Tạo lập một đề cương sơ lược về kiến trúc được bắt đầu với các phần mềm nền, với những hệ khác thuộc môi trường mà hệ thống tương tác với chúng. Tiếp theo, người kiến trúc phải làm việc với một tập con các ca sử dụng đã xác định thể hiện những chức năng chính yếu của hệ thống . Mỗi ca sử dụng được lựa chọn là sự cụ thể hóa đặc trưng của kiến trúc và được thực hiện dưới dạng các hệ thống con, các lớp và các thành phần chủ yếu.

6.2.2. Phân tích kiến trúc

Phân tích kiến trúc bao gồm việc chính là *xác định các gói, các lớp thực thể hiển nhiên*. Ngoài ra người ta còn xác định các yêu cầu chuyên biệt chung nhất.

6.2.2.1. Xác định các gói

Việc xác định các gói bao gồm:

- Xác định các gói phân tích
- Xác định các gói dịch vụ
- Xác định mối quan hệ phụ thuộc giữa các gói phân tích

6.2.2.1.1. Xác định các gói phân tích

Từ yêu cầu chức năng nghiệp vụ, xác định các gói phân tích. Đó là phương tiện tổ chức hệ thống thành các "cụm" nhỏ hơn cho dễ quản lý. Một gói phân tích có thể chứa các *lớp phân tích, các thực thi ca sử dụng* và các gói phân tích khác.

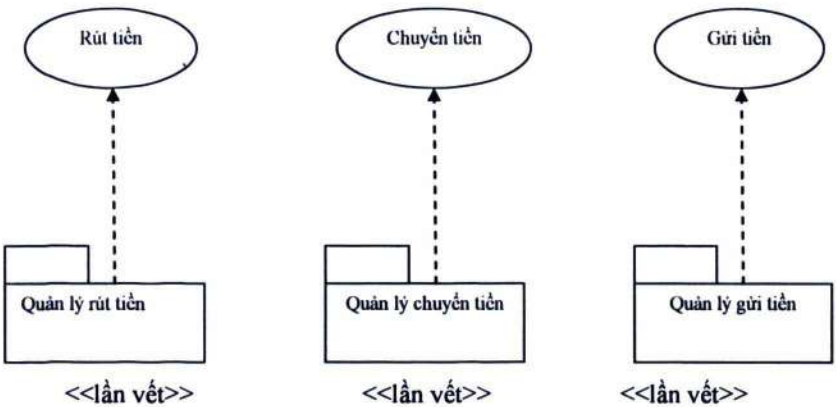
Hai bước xác định các gói phân tích:

Bước 1: bố trí một số ca sử dụng vào các gói riêng (có thể đã thực hiện ở bước xác định nhu cầu) theo một số tiêu chí gợi ý sau:

- Các ca sử dụng cần có để hỗ trợ một quá trình nghiệp vụ cụ thể;
- Các ca sử dụng cần có để hỗ trợ một tác nhân cụ thể của hệ thống;
- Các ca sử dụng có quan hệ với nhau bằng các quan hệ tổng quát hoá, mở rộng và quan hệ bao gồm.

Bước 2: tiến hành thực thi chức năng tương ứng bên trong gói đó

Ví dụ 6.1:

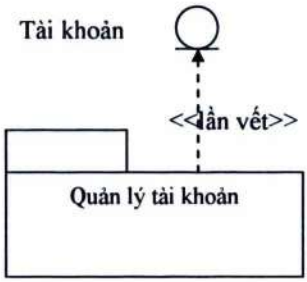


Hình 6.7. Xác định các gói phân tích riêng biệt từ các ca sử dụng trong cùng một gói (cùng gói với chúng có liên quan đến cùng một quá trình nghiệp vụ thanh toán tín dụng)

Chú ý: Trong nhiều trường hợp, ta có thể tìm thấy các phần chung trong các gói phân tích, chẳng hạn như một lớp phân tích nào đó. Để xử lý vấn đề này, ta đặt phần chung đó vào một gói riêng nằm ngoài các gói chứa nó, sau đó để các gói khác có liên quan phụ thuộc vào gói mới chứa lớp chung này. Các lớp có phần chia sẻ chung như vậy thường là các lớp thực thể. Chúng có thể tìm thấy bằng cách lần vết tới các lớp thực thể lĩnh vực hoặc nghiệp vụ. Do vậy ta nên nghiên cứu các lớp thực thể lĩnh vực hay lớp thực thể nghiệp vụ để tìm ra phần chung và tạo nên gói phân tích tổng quát.

Ví dụ 6.2:

Trong hệ thống giao dịch tín dụng cả 3 gói rút tiền, chuyển tiền và gửi tiền đều có một lớp chung là lớp thực thể tài khoản. Lớp này là đại diện của lớp thực thể miền quan trọng là tài khoản và chúng được chia sẻ trong các gói phân tích nêu trên. Do vậy, ta cần tạo ra một gói riêng cho nó là gói quản lý tài khoản:



Hình 6.8. Gói quản lý tài khoản là gói chung được xác định từ một lớp miền

6.2.2.1.2. Xác định các gói dịch vụ

Xác định các gói dịch vụ là trường hợp riêng của việc xác định gói. Công việc này thích hợp sau khi các yêu cầu chức năng đã được hiểu rõ và đã xác định được phần lớn các lớp phân tích.

Gói dịch vụ dùng để mô tả các gói phân tích được sử dụng ở một mức thấp hơn trong sơ đồ phân cấp cấu trúc các gói của hệ thống.

Một gói dịch vụ có thể có các tính chất sau:

- Chứa một tập hợp các lớp có liên quan với nhau về mặt chức năng;
- Không thể chia nhỏ hơn;
- Có thể tham gia vào một hay nhiều thực thi ca sử dụng;
- Phụ thuộc rất ít vào các gói dịch vụ khác;
- Các chức năng mà nó cung cấp có thể được quản lý như một đơn vị riêng biệt.

Ví dụ 6.3

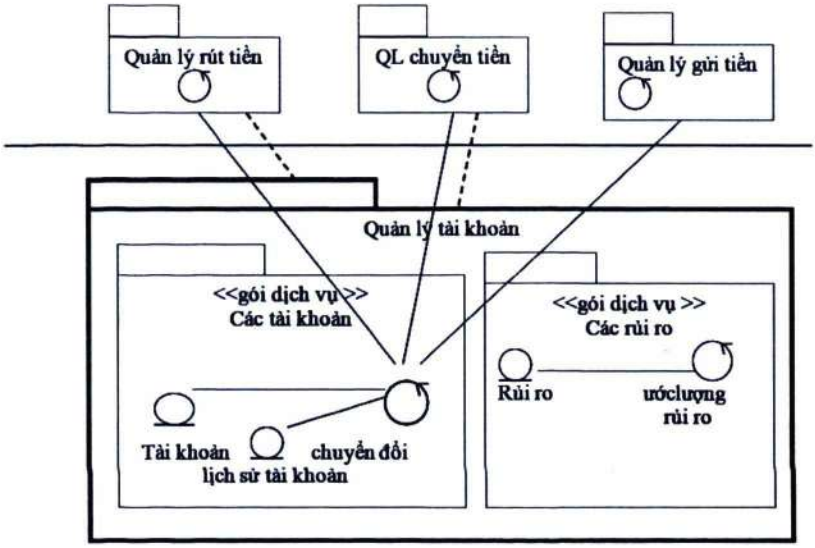
Gói chung quản lý tài khoản chính là gói dịch vụ:

Các bước xác định các gói dịch vụ:

- Xác định một gói dịch vụ cho mỗi dịch vụ được chọn.

- Xác định một gói dịch vụ cho mỗi dịch vụ mà có thể trở thành tuỳ chọn cho nhiều ca sử dụng. Ta sẽ xác định một gói dịch vụ cho mỗi dịch vụ được cung cấp bởi các lớp có liên quan về mặt chức năng.

6.2.2.1.3. Xác định mối quan hệ phụ thuộc giữa các gói phân tích



Hình 6.9. Xác định các gói dịch vụ gói các lớp có liên quan về chức năng

Mục tiêu đặt ra là tìm các gói phân tích tương đối độc lập với các gói khác, tức là chúng được ghép nối lỏng lẻo với nhau nhưng có tính kết dính cao bên trong. Với mục tiêu này, cần cố gắng giảm số lượng các mối quan hệ giữa các lớp thuộc các gói khác nhau.

Cách tốt nhất để thực hiện công việc này là tổ chức mô hình phân tích thành các tầng và xếp các gói ứng dụng cụ thể ở tầng đỉnh hay tầng ứng dụng, các gói chung hơn ở tầng thấp hơn như hình 6.9.

Các gói trên tầng ứng dụng đều chứa một lớp điều khiển (điều khiển rút tiền, điều khiển chuyển tiền và điều khiển gửi tiền). Các lớp điều khiển này đều liên kết (đường nối liền) với lớp điều khiển chuyển đổi trong gói quản lý tài khoản. Do đó đòi hỏi phải có mối quan hệ phụ thuộc (mũi tên có đường đứt đoạn) tương ứng giữa 3 gói trên tầng ứng dụng đó với gói quản lý tài khoản.

6.2.2.1.4. Phân tích một gói

Mục đích của việc phân tích một gói:

- Bảo đảm rằng gói phân tích càng độc lập đối với các gói khác nếu có thể;

- Bảo đảm rằng gói phân tích hoàn thành mục đích của nó là thực thi những lớp miền hoặc các ca sử dụng nào đó;

- Mô tả các mối quan hệ phụ thuộc sao cho có thể ước tính được hiệu ứng của các thay đổi này.

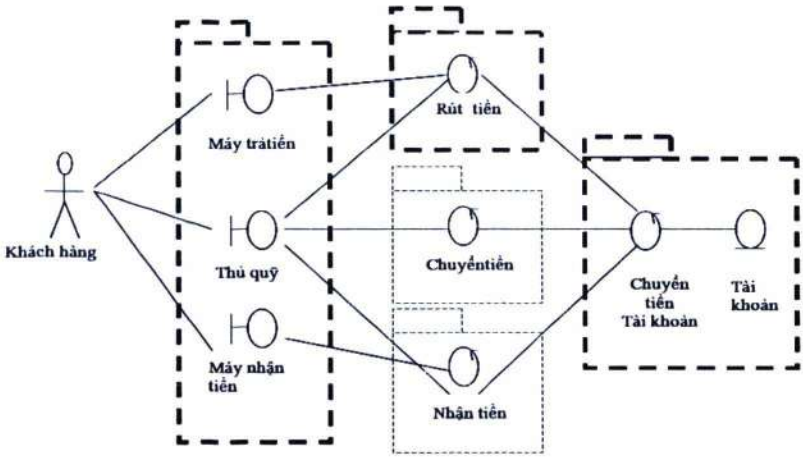
Một số nguyên tắc chung phân tích thành gói:

- Xác định và duy trì các mối quan hệ phụ thuộc giữa hai gói có chứa các lớp liên kết với nhau;

- Mỗi gói chứa các lớp đúng, nghĩa là hãy cố gắng làm cho gói trở thành kết dính bằng cách chỉ đa các đối tượng có liên quan về mặt chức năng vào trong gói;

- Hạn chế tối đa các mối quan hệ phụ thuộc tới các gói khác bằng cách bố trí lại các lớp chứa trong một gói sang gói khác nếu nó quá phụ thuộc vào các gói khác.

Ví dụ 6.4:



Hình 6.10. Mô hình lớp phân tích của hệ thống giao dịch tín dụng với các gói

(đường đứt nét là hình ảnh các gói. Các gói có đường viền đậm là những gói có tầm quan trọng về mặt kiến trúc vì chúng liên quan đến ca sử dụng rút tiền là ca sử dụng quan trọng nhất).

6.2.2.2. Xác định các lớp thực thể hiển nhiên

Việc xác định các lớp thực thể quan trọng nhất dựa trên các lớp miền hoặc các thực thể nghiệp vụ đã được xác định trong quá trình nắm bắt các yêu cầu. Mỗi lớp thực thể này có thể đưa vào một gói riêng. Ở bước này không nên xác định quá nhiều lớp và đi quá sâu vào chi tiết. Một phác thảo ban đầu chỉ gồm các lớp có ý nghĩa về mặt kiến trúc là đủ.

6.2.2.3. Xác định các yêu cầu chuyên biệt chung nhất

Yêu cầu chuyên biệt là yêu cầu nảy sinh trong quá trình phân tích và việc nắm bắt nó là quan trọng. Các yêu cầu này có thể là:

- Tính lâu bền (cần lưu trữ);

- Phân tán đối tượng trong suốt (các đối tượng được phân tán nhưng ngời sử dụng không cần biết nó được cư trú ở đâu);

- Sự phân bố và tính tương tranh;

- Các đặc trưng về an toàn;

- Phát hiện lỗi, khắc phục lỗi, dung sai về lỗi;

- Quản lý giao dịch;

...

Các yêu cầu đặc biệt chung có thể được tham chiếu tới như là các yêu cầu đặc biệt của các thực thi ca sử dụng cụ thể và các lớp phân tích.

6.2.3. Thiết kế kiến trúc

Thiết kế kiến trúc bao gồm việc thiết kế các cấu hình mạng, thiết kế các hệ thống con và các giao diện của chúng, xác định các lớp quan trọng về mặt kiến trúc.

6.2.3.1. Thiết kế cấu hình mạng cho hệ thống

Các cấu hình mạng chung thường dùng một dạng mẫu ba hay hai tầng (three/tier pattern) trong đó các ứng dụng khách hàng được phân vào một tầng, chức năng CSDL vào một tầng, và logic nghiệp vụ/ứng dụng vào một tầng. Dạng đơn giản của kiến trúc *Client-Server* là một trường hợp đặc biệt của dạng mẫu 2 tầng, trong đó logic nghiệp vụ/ ứng dụng được bố trí vào cùng một tầng với tầng *client* hoặc tầng *CSDL*.

Kiến trúc phổ biến chung hiện nay là kiến trúc ba tầng: tầng giao diện, tầng tác nghiệp và tầng lưu trữ:

1. *Tầng trình diễn/tầng giao diện*: biểu diễn và giới thiệu các thành phần của hệ thống thông qua các giao diện đồ hoạ, các Window, các hộp thoại,... Các thực thể trong hệ thống được thể hiện sao cho vừa thân thiện với NSD, vừa phù hợp với bài toán ứng dụng.

2. *Tầng logic ứng dụng/tầng tác nghiệp*: mô tả các đối tượng thực thi các nhiệm vụ và các quy luật điều hành các tiến trình của hệ thống. Hệ thống phần mềm có hai loại đối tượng cơ bản:

- *Những đối tượng nghiệp vụ:* là những đối tượng đại biểu cho các khái niệm của miền ứng dụng.

- *Những đối tượng dịch vụ* (*Service Object*): những đối tượng không nằm trong phạm vi bài toán nhưng cung cấp các dịch vụ cho hệ thống như: làm môi giới, tương tác với CSDL, trao đổi thông tin, lập báo cáo, đảm bảo an toàn, an ninh dữ liệu, ...

Tầng hai được thể hiện ở các sơ đồ: lớp, trạng thái, cộng tác, hoạt động và sơ đồ triển khai.

3. *Tầng lưu trữ* (*Storage*): thể hiện cơ chế lưu trữ đảm bảo nhất quán và bền vững dữ liệu. Hiện nay có hai mô hình CSDL chính đang được sử dụng phổ biến là: *mô hình dữ liệu quan hệ và mô hình dữ liệu hướng đối tượng.*

Để lưu trữ dữ liệu cho hệ thống, ta phải lựa chọn hệ quản trị CSDL và những phương pháp biến đổi dữ liệu, biến đổi các câu truy vấn cho phù hợp với công nghệ hiện đại và với phạm vi ứng dụng. Kiến trúc sư đối tượng phải lựa chọn công nghệ CSDL thích hợp để phát triển hệ thống.

Khi phân tích, thiết kế hướng đối tượng và nếu lựa chọn mô hình dữ liệu quan hệ thì người phát triển hệ thống phải quan tâm đến những vấn đề về tích hợp dữ liệu để đảm bảo cho phép người sử dụng truy cập được tới tất cả các dữ liệu từ nhiều mô hình khác nhau một cách trong suốt.

Quá trình phát triển phần mềm cũng giống như quá trình học và nhận thức của con người, đó là quá trình lặp, tích luỹ để phát triển, hoàn thiện liên tục. Vì vậy, kiến trúc của hệ thống cũng phải được xây dựng sao cho phù hợp với sự phát triển và khả năng mở rộng của hệ thống.

Trong triển khai cài đặt, kiến trúc ba tầng thường được tổ chức theo nhiều phương án khác nhau. Có thể thực hiện:

- Cả 3 tầng trên cùng máy (hệ thống nhỏ);

- Tầng trình diễn và tầng logic ứng dụng trên máy khách (*Client Computer*), tầng lưu trữ trên Sever (hệ thống loại vừa);

- Tầng trình diễn trên máy người sử dụng, tầng logic ứng dụng trên máy trạm phục vụ ứng dụng (*Application Server*), còn tầng lưu trữ tổ chức ở trạm phục vụ dữ liệu (Data Server) (những hệ thống lớn).

Hiện nay có những ngôn ngữ lập trình hướng đối tượng như Java và các công nghệ CSDL hướng đối tượng đã sẵn sàng hỗ trợ cài đặt phân tán theo hai phương án cuối.

Những đặc trưng của các cấu hình mạng bao gồm:

- Những nút nào liên kết với nhau, các khả năng về công suất xử lý và kích cỡ bộ nhớ của chúng là bao nhiêu?

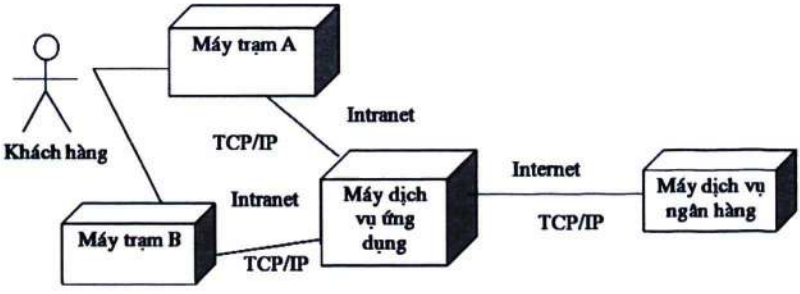
- Kết nối giữa các nút thuộc loại nào, các giao thức truyền thông giữa chúng là gì?

- Các đặc trưng của các kết nối và các giao thức truyền thông (băng thông, tính sẵn sàng, chất lượng dịch vụ của nó)?

- Nhu cầu về khả năng xử lý dư thừa, về chế độ hỏng hóc, về sự di trú tiến trình xử lý, về việc duy trì các bản sao lưu dữ liệu dự phòng,...?

Ví dụ 6.5

Cấu hình mạng cho hệ thống:



Hình 6.11.Biểu đồ bố trí các nút của hệ thống

6.2.3.2.Thiết kế các hệ thống con và các giao diện của chúng

Mục tiêu thiết kế một hệ thống con bao gồm:

- Bảo đảm cho một hệ thống con là độc lập đối với các hệ thống con khác đến mức tối đa có thể được thông qua các giao diện của chúng;

- Bảo đảm rằng các hệ thống con cung cấp các giao diện đúng;

- Bảo đảm rằng hệ thống con thực thi đúng các thao tác đã được xác định bởi các giao diện mà nó cung cấp.

Việc phân hệ thống thành các hệ thống con là một cách thức để tổ chức mô hình thiết kế thành các cụm có thể quản lý được. Hơn nữa, việc này giúp chúng ta có thể lập luận khi thiết kế và đánh giá được các cơ hội tái sử dụng.

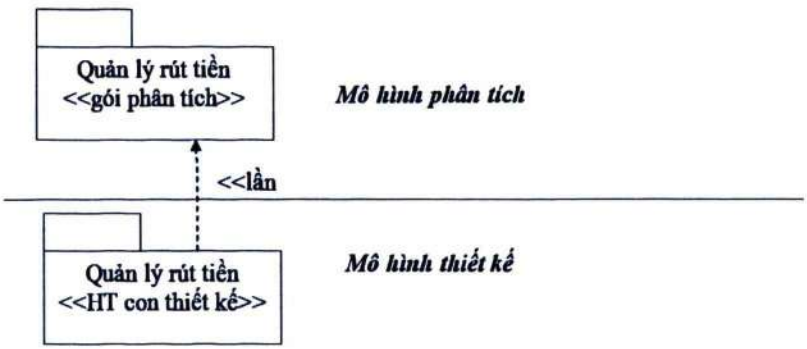
6.2.3.2.1. Xác định các hệ thống con

Chúng ta cần xác định các hệ thống con trong các tầng ứng dụng cụ thể và tầng ứng dụng tổng quát (2 tầng mức đỉnh).

Để xác định các hệ thống con, người ta thường sử dụng các gói phân tích đã tìm thấy trong quá trình phân tích. Tất nhiên sau này cần tinh chế lại.

Hệ thống con thiết kế có thể bao gồm các lớp thiết kế, các thực thi ca sử dụng, các giao diện và các hệ thống con khác. Hệ thống con có thể cung cấp các giao diện thể hiện các chức năng xuất ra từ nó dưới dạng các tác vụ mà nó cung cấp.

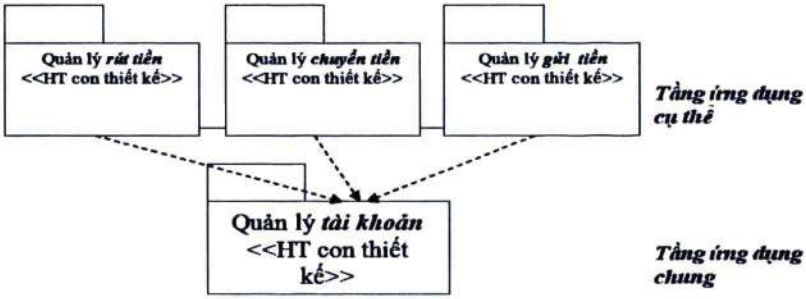
Ví dụ 6.6:



Hình 6.12. Xác định các hệ thống con dựa trên các gói phân tích đã có

Chúng ta cần tinh chế các hệ thống con để xử lý các chức năng dùng chung.

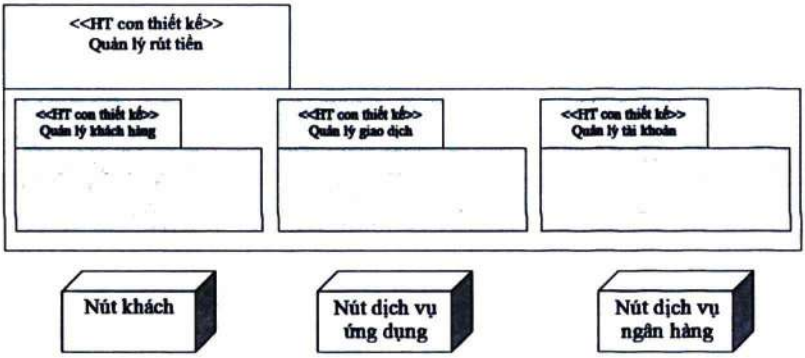
Ví dụ 6.7:



Hình 6.13. Hệ thống con dịch vụ Quản lý tài khoản cung cấp một dịch vụ tổng quát có thể được sử dụng trong nhiều ca sử dụng khác nhau

Các lớp thiết kế thực hiện ca sử dụng cần được phân phối cho nhiều nút khác nhau khi bố trí để phục vụ các tác nhân. Vì thế các lớp của nó cần được phân vào nhiều thành phần nhỏ hơn để có thể bố trí chúng một cách thích hợp vào các nút tương ứng.

Ví dụ 6.8:



Hình 6.14. Phân bố các hệ thống con trên các nút của mạng dựa vào việc phân các lớp thiết kế thực hiện ca sử dụng rút tiền vào 3 thành phần nhỏ hơn

Một hệ thống con hoàn thành mục tiêu của nó nếu nó cung cấp một thực thi đúng các thao tác xác định bởi giao diện mà nó cung cấp. Một số vấn đề liên quan ở đây là:

- Mỗi giao diện do hệ thống con cung cấp phải được các lớp thiết kế hoặc các hệ thống con khác bên trong hệ thống con đó cung cấp phương tiện thực thi;

- Để làm sáng tỏ cách thức mà các thiết kế bên trong của một hệ thống con thực thi bất kỳ một giao diện nào hoặc các ca sử dụng của nó, ta có thể tạo ra các cộng tác dưới dạng các yếu tố được chứa trong hệ thống con đó.

6.2.3.2.2. Xác định các mối quan hệ giữa các hệ thống con

Sử dụng các mối quan hệ phụ thuộc giữa các gói phân tích làm cơ sở xác định các mối quan hệ của các hệ thống con thiết kế. Các mối quan hệ phụ thuộc nên được biểu diễn tường minh giữa các tầng của kiến trúc.

Các mối quan hệ phụ thuộc phải được xác định và duy trì từ hệ thống con này đến hệ thống con khác có chứa các phần tử được liên kết với nó. Tuy nhiên, nếu các hệ thống con khác đó cung cấp các giao diện thì các mối quan hệ phụ thuộc cần được khai báo hướng về các giao diện đó. Một phụ thuộc tốt là phụ thuộc vào một giao diện mà không phụ thuộc vào hệ thống con. Ta nên tối thiểu hoá các phụ thuộc vào các hệ thống con và/hoặc các giao diện bằng việc bố trí lại các lớp được chứa mà không quá phụ thuộc vào các hệ thống con khác.

6.2.3.2.3. Xác định các giao diện của các hệ thống con

Các giao diện mà một hệ thống con cung cấp xác định các thao tác mà từ "bên ngoài" hệ thống con đó có thể truy nhập đến nó. Các giao diện này do các lớp hoặc các hệ thống con khác bên trong hệ thống con đó cung cấp.

Giao diện sử dụng các đặc tả các tác vụ mà các lớp thiết kế và các hệ thống con cung cấp. Phần lớn các giao diện giữa các hệ thống con đều mang ý nghĩa về mặt kiến trúc, xác định phạm vi và cách thức mà các hệ thống con được phép tương tác với nhau.

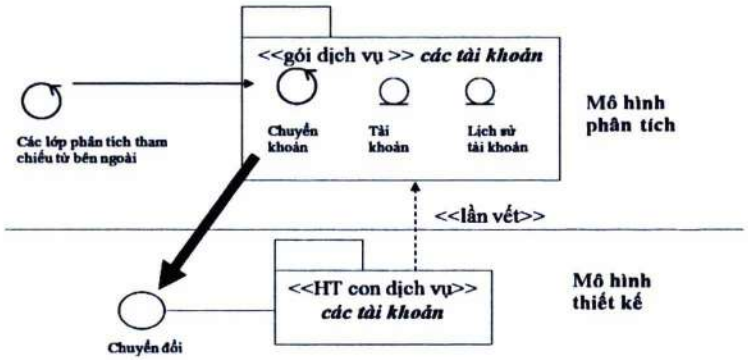
Các thao tác được xác định qua các giao diện được cung cấp bởi một hệ thống con cần phải hỗ trợ mọi vai trò mà hệ thống con này đóng góp trong thực thi các ca sử dụng khác nhau. Ngay cả khi các giao diện đã được phác thảo, các giao diện này có thể phải tinh chế khi phát triển mô hình thiết kế. Một hệ thống

con và các giao diện của nó có thể được sử dụng bên trong nhiều thực thi ca sử dụng, do đó sẽ cung cấp nhiều yêu cầu mới nữa trên các giao diện.

Ví dụ 6.9:

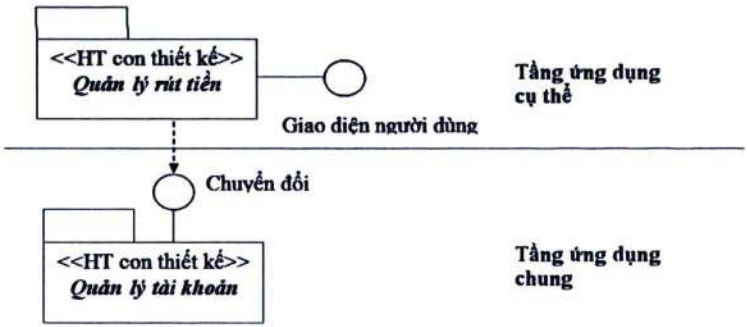
Tìm các giao diện của hệ con dựa trên mô hình phân tích.

Trong mô hình thiết kế chúng ta có thể xác định một giao diện ban đầu của hệ thống con dịch vụ gọi là ***các tài khoản*** do nó cung cấp:



Hình 6.15. Giao diện của một hệ thống con được xác định từ gói thiết kế tương ứng

Tương tự ta xác định được các giao diện của các hệ thống con thiết kế khác thuộc hai tầng ứng dụng và các tham chiếu của các hệ thống con khác đến các giao diện này.



Hình 6.16. Giao diện của các hệ thống con thuộc các tầng ứng dụng

6.2.3.3. Xác định các lớp thiết kế quan trọng về mặt kiến trúc

6.2.3.3.1. Xác định các lớp thiết kế từ các lớp phân tích

Trong quá trình phân tích, người ta xác định được các lớp phân tích quan trọng về mặt kiến trúc, từ đó phác thảo được một số lớp thiết kế. Các mối quan hệ giữa các lớp phân tích quan trọng này được dùng để xác định các mối quan hệ giữa các lớp thiết kế tương ứng.

6.2.4.3.2. Xác định các lớp hoạt động

Các lớp hoạt động do hệ thống yêu cầu có thể được xác định bằng cách xem xét các yêu cầu đồng thời đối với hệ thống, chẳng hạn như:

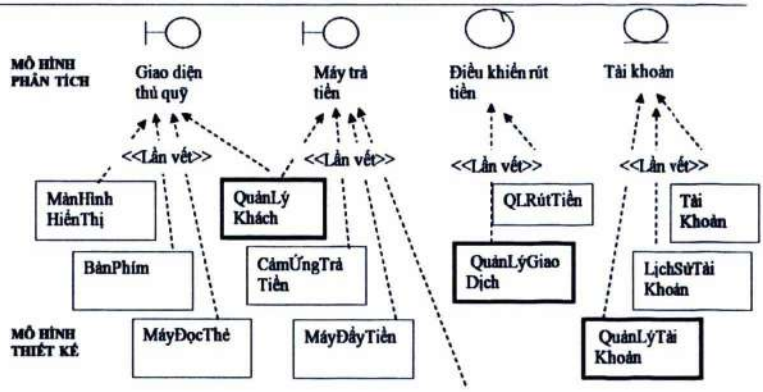
- Các yêu cầu về hiệu năng, lưu lượng thông qua, về tính sẵn sàng của hệ thống mà các tác nhân yêu cầu cần có khi tương tác với hệ thống. Chẳng hạn, nếu có một tác nhân nào đó có các yêu cầu cao về thời gian phúc đáp, thì yêu cầu đó được quản lý bằng một đối tượng hoạt động dành riêng để nhận đầu vào từ tác nhân và cung cấp đầu ra cho tác nhân đó.
- Sự phân bố hệ thống trên các nút. Các đối tượng hoạt động cần hỗ trợ bằng sự phân phối trên nhiều nút khác nhau
- Các yêu cầu khác như là các yêu cầu về sự khởi động và kết thúc hệ thống, tính sống động, việc tránh bế tắc, tránh thiếu tài nguyên, tái cấu hình các nút và năng lực kết nối.

Để phác thảo các lớp hoạt động ban đầu có thể sử dụng các kết quả phân tích và mô hình triển khai làm đầu vào rồi sau đó bố trí các thiết kế tương ứng của các lớp phân tích (hoặc các bộ phận của chúng) cho các nút thông qua các lớp hoạt động.

Một khả năng khác để phác thảo các lớp hoạt động là sử dụng các hệ thống con được xác định trước đó và phân toàn bộ một hệ thống con cho một nút riêng bằng cách xác định một lớp hoạt động bên trong hệ thống con.

Ví dụ 6.10:

Tìm các lớp thiết kế từ các lớp phân tích khi tinh chế nó để phù hợp với yêu cầu của môi trường triển khai:



Hình 6.17. Các lớp thiết kế nhận được từ các lớp phân tích khi làm mịn

Phác thảo các lớp hoạt động: trước đây, chúng ta đã xác định các lớp phân tích như lớp *quản lý khách, quản lý giao dịch (rút tiền), tài khoản*. Bây giờ, chúng ta nhận thấy khách hàng quan tâm tới chức năng được cung cấp bởi lớp phân tích *quản lý khách*. Còn việc xử lý yêu cầu là nhiệm vụ của lớp quản lý rút tiền và lớp quản lý tài khoản đã được tách riêng đặt trên các máy dịch vụ. Do vậy, ta xác định được các lớp hoạt động như các lớp có hình viền đậm trên hình vẽ.

6.2.4. Kiến trúc đối tượng phân tán

6.2.4.1. Khái niệm về đối tượng phân tán và kiến trúc đối tượng phân tán

Đối tượng phân tán (distributed object) cũng là một hướng phát triển tự nhiên từ phương pháp luận hướng đối tượng. Trong khi kỹ nghệ hướng thành phần quan tâm tới việc đóng gói các đối tượng để có thể sử dụng lại một cách thuận tiện, thì kỹ nghệ đối tượng phân tán tập trung vào vấn đề kiến trúc các đối tượng có khả năng giao tiếp một cách trong suốt trên các nền và hệ thống mạng khác nhau (giao tiếp ngầm). Cũng giống như hướng thành phần, một đối tượng phân tán có thể thực hiện ở một ngôn ngữ bất kỳ, nhưng nó phải có các giao diện theo một chuẩn nào đó để có thể hợp tác với nhau liên quá trình và xuyên mạng một cách đơn giản như hai đối tượng trong một chương trình. Như vậy, một đối tượng phân tán cũng được sử dụng khi đã biên dịch, đóng gói hoàn chỉnh dưới dạng một server. Có thể nêu ra một số ví dụ về đối tượng phân

tán ở đây như Web-Server, OPC-Server, XML-Services. Tuy nhiên, việc sử dụng chúng có thể vẫn đòi hỏi phải lập trình phía client (một đối tượng phân tán hoặc một chương trình ứng dụng thông thường). Ngày nay, kỹ nghệ hướng đối tượng phân tán và hướng thành phần đã gặp nhau ở nhiều điểm, ví dụ trong công nghệ COM/DCOM/ActiveX [27].

Một đối tượng phân tán là một đối tượng phần mềm trong một hệ thống phân tán, có thể được sử dụng bởi các chương trình ứng dụng hoặc các đối tượng khác thuộc cùng một quá trình tính toán, thuộc một quá trình tính toán khác hoặc thuộc một trạm khác trong mạng theo một phương thức thống nhất thông qua giao tiếp ngầm (không để ý tới giao thức truyền thông cụ thể, trong suốt với hệ điều hành, kiến trúc phần cứng và hệ thống mạng).

Một đối tượng phân tán có các thuộc tính có thể truy cập được từ xa, có các phép toán có thể gọi được từ xa.

Mỗi đối tượng phân tán (bất kể dạng thực hiện, nền triển khai và vị trí cài đặt) đều có căn cước phân biệt và có thể được sử dụng như các đối tượng nội trình (in-process object). Lợi thế quyết định ở đây là việc tạo dựng một ứng dụng phân tán được thực hiện ở mức trừu tượng cao hơn so với kiểu lập trình mạng cổ điển, nhờ vậy trên nguyên tắc không khác biệt so với tạo dựng một ứng dụng đơn độc (stand-alone application). Để đạt được điều đó, ta cần sự hỗ trợ hữu hiệu của một phần mềm khung (framework).

Hiện nay có hai mô hình chuẩn cho những công trình khung là COM/DCOM (*Component Object Model/Distributed Component Object Model*, gọi chung là COM) và CORBA(*Common Object Request Broker Architecture*). CORBA cho phép sử dụng một cách rộng rãi và linh hoạt hơn, trong khi DCOM hiện nay hầu như chỉ sử dụng được trên các hệ Microsoft Windows (95, 98, NT, 2000, XP).

Một kiến trúc đối tượng phân tán định nghĩa mô hình các đối tượng phân tán, mô hình giao tiếp và chuẩn giao tiếp giữa các đối tượng phân tán.

6.2.4.2. Các đặc tính chung của một kiến trúc đối tượng phân tán

Một kiến trúc đối tượng phân tán tạo điều kiện cho việc lập trình ở một mức trừu tượng cao hơn so với phương pháp hướng đối tượng cổ điển. Cụ thể,

điểm khác biệt so với lập trình hướng đối tượng cổ điển nằm ở "*tính trong suốt phân tán*" (*distribution transparency*), thể hiện qua các đặc tính sau:

1-*Trong suốt vị trí*: Một Client không cần biết rằng đối tượng server nằm trong cùng một quá trình tính toán, thuộc một quá trình tính toán khác trên cùng một trạm hoặc trên một trạm khác, cách sử dụng đối tượng server là thống nhất. Ngược lại cũng vậy;

2-*Trong suốt thể hiện*: Client không cần quan tâm tới việc đối tượng server được thể hiện bằng ngôn ngữ lập trình nào và bằng phương pháp nào, mà chỉcần quan tâm tới giao diện để có thể sử dụng;

3-*Trong suốt nền*: Một Client không cần biết rằng đối tượng Server nằm trên hệ điều hành nào, trên nền máy tính kiến trúc ra sao;

4-*Trong suốt truyền thông*: Mã thực hiện Client và các đối tượng Server không liên quan tới mạng truyền thông và giao thức truyền thông cụ thể·

Để đáp ứng các yêu cầu trên, người ta thường áp dụng các mẫu thiết kế sau đây:

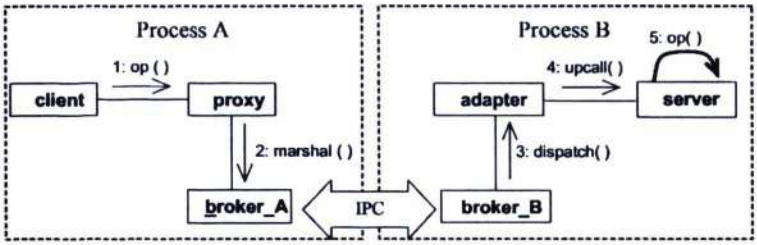
- Proxy: Một đối tượng đại diện cho Server bên phía Client, để Client có thể sử dụng đối tượng Server đơn giản như với một đối tượng nội trình (ví dụ thông qua con trỏ).

- Broker: Bộ phận che dấu chi tiết về cơ chế truyền thông cụ thể, tạo điều kiện cho Client/proxy và Server giao tiếp với nhau mà không phụ thuộc vào nền và hệ thống truyền thông bên dưới. Broker có mặt cả bên Client và Server.

- Adapter: Đối tượng trung gian, có vai trò thích ứng giao diện giữa broker và Server, tạo điều kiện cho việc phát triển Server một cách độc lập, cũng như sử dụng các Server có sẵn (chưa tuân theo kiến trúc đối tượng phân tán).

- Marshaling/Unmarshaling: Cơ chế thực hiện mã hóa và đóng gói các lời gọi hàm bên Client thành các bức điện tương ứng với cơ chế truyền thông cấp thấp, cũng mở gói và giải mã các bức điện thành lời gọi hàm chi tiết bên đối tượng Server. Các phương thức tương tựcũng được thực hiện để chuyển kết quả từ Server trở lại Client. Các nhiệm vụ này do proxy, Server hoặc/và adapter đảm nhiệm.

- Interface Mapping: Giải quyết vấn đề trong suốt thể hiện bằng cách mô tả các giao diện bằng một ngôn ngữ độc lập IDL (Interface Description Language) và cho phép ánh xạ sang thực hiện bằng một ngôn ngữ cụ thể.



Hình 6.18. Các mẫu thiết kế giao tiếp giữa Client và đối tượng Server [26]

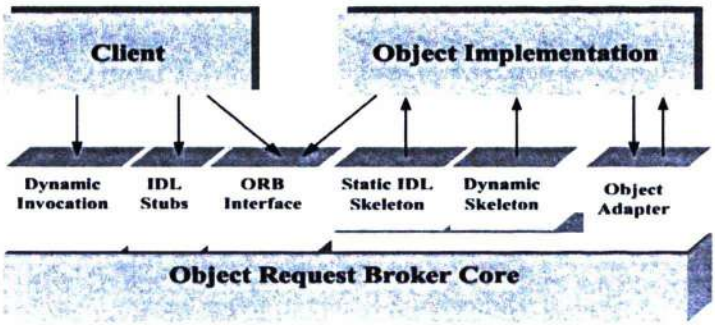
6.2.4.3. Chuẩn CORBA

Chuẩn CORBA do tổ chức OMG (Object Management Group) quản lý và phát triển. Đây là hiệp hội lớn nhất của các nhà phát triển, sản xuất và ứng dụng phần mềm trên thế giới.

Chuẩn CORBA đưa ra một kiến trúc đối tượng phân tán cùng với các đặc tả ứng dụng cho nhiều lĩnh vực khác nhau, nhiều nền khác nhau và nhiều ngôn ngữ lập trình khác nhau.

Vì tính trung lập của nó, CORBA được hỗ trợ rất rộng rãi, đặc biệt trong các hệ thống thông tin thương mại, phần mềm giao dịch kinh doanh và dịch vụ viễn thông. Tuy nhiên, cũng do tính độc lập của nó dẫn đến nhiều lý do mà CORBA không thực sự mạnh ở các hệ thống ứng qui mô vừa và nhỏ.

Hình 6.19 minh họa cấu trúc mô hình CORBA, trong đó bộ phận trừu tượng trung gian mang tên Object Request Broker (ORB) giữ vai trò quan trọng nhất. ORB cho phép khách hàng (Client) sử dụng dịch vụ của đối tượng phục vụ (Server) mà không cần biết cụ thể dạng thực hiện, nền triển khai và vị trí cài đặt của đối tượng phục vụ. Kiến trúc ở đây được thực hiện theo các mẫu thiết kế đã trình bày trong phần trước.



Hình 6.19. Cấu trúc mô hình CORBA [26]

6.2.4.4. Chuẩn COM/DCOM

COM (Component Object Model) là một mô hình đối tượng thành phần, một mô hình cơ sở cho nhiều công nghệ phần mềm quan trọng của hãng Microsoft. COM định nghĩa chuẩn nhị phân và đặc tả kết nối cho việc tương tác giữa các thành phần của một phần mềm với một thành phần khác trên cùng một quá trình tính toán, trên nhiều quá trình khác nhau hay trên các máy tính riêng biệt. Hãng Microsoft cũng hy vọng một ngày không xa COM cũng được sử dụng phổ biến trên các nền phần cứng và hệ điều hành khác nhau.

COM là một mô hình lập trình cơ sở đối tượng thiết kế để nâng cao sự tương tác giữa các thành phần phần mềm, nghĩa là, cho phép hai hoặc nhiều ứng dụng hay các thành phần dễ dàng giao tiếp với nhau cho dù chúng được viết bởi nhiều người khác nhau trong những khoảng thời gian khác nhau, bằng nhiều ngôn ngữ lập trình khác nhau thậm chí chạy trên các máy tính khác nhau, không hay cài đặt cùng một hệ điều hành. Để thực hiện điều này, COM định nghĩa và thực thi các kỹ thuật cho phép các ứng dụng kết nối với nhau như các đối tượng phần mềm.

Nói cách khác, COM đưa ra một mô hình tương tác mà qua đó một khách hàng (Client) có thể kết nối với các nhà cung cấp dịch vụ (object) đó một cách thuận tiện. Với COM, các ứng dụng kết nối với nhau và với hệ thống qua các tập hợp của các lời gọi hàm (xem như là các phương thức hay những hàm thành viên) trên giao diện.

Theo cách tư duy COM, một giao diện là một "quy ước" kiểu mạnh giữa các thành phần phần mềm nhằm cung cấp những liên quan dù nhỏ nhưng hữu dụng tập các thao tác liên quan danh nghĩa. Một đối tượng được định nghĩa phù hợp với COM là một sự thể hiện đặc biệt của đối tượng. Một đối tượng COM giống như một đối tượng C++ nhưng khác ở chỗ một Client không truy nhập trực tiếp vào đối tượng COM mà sẽ qua các giao diện do đối tượng cung cấp.

6.2.4.4.1. Giao diện

Cách duy nhất để truy cập dữ liệu hoặc tác động lên một đối tượng COM là thông qua giao diện của nó. Một giao diện thực chất là một nhóm các hàm có sẵn liên quan với nhau. Có thể so sánh một giao diện với một lớp cơ sở trừu tượng chỉ gồm các hàm thuần ảo trong ngôn ngữ C++. Giao diện định nghĩa cú pháp các hàm thành viên, gọi là các phương thức (methods), kiểu trả về, số lượng và các kiểu tham số. Một giao diện không qui định cụ thể các phương thức đó được thực hiện như thế nào. Thực chất việc thể hiện giao diện là sử dụng con trỏ truy nhập vào một mảng các con trỏ khác và các con trỏ này trỏ tới các hàm của giao diện.

Thông thường, tên của giao diện được bắt đầu bằng chữ cái I, ví dụ như IUnknown, IData... Định danh thật của giao diện thể hiện ở chỉ danh GUID của nó, còn tên chỉ để thuận tiện cho việc lập trình và hệ thống COM sẽ sử dụng các chỉ danh này khi thao tác trên giao diện.

Khi giao diện có tên hoặc kiểu cụ thể và tên của các hàm thành viên, nó chỉ định nghĩa làm thế nào một Client có thể sử dụng giao diện đó và những đáp ứng mong đợi từ đối tượng qua giao diện đó. Ví dụ, giao diện IStack với hai hàm thành viên PUSH và POP chỉ định nghĩa những thông số và kiểu trả về của hai hàm này và những gì chúng được mong đợi thực hiện từ Client.

Có thể nói, giao diện là phương tiện để đối tượng đưa ra những dịch vụ của nó. Có bốn điểm quan trọng về giao diện cần chú ý:

1- Một giao diện không phải là một lớp theo định nghĩa lớp thông thường bởi một lớp có thể được thể hiện qua một đối tượng còn một giao diện thì không bởi nó không kèm theo sự thực thi.

2- Một giao diện không phải là một đối tượng bởi một giao diện chỉ đơn thuần là một nhóm các hàm liên quan và là chuẩn nhị phân mà qua đó Client và object có thể giao tiếp với nhau. Còn đối tượng thì có thể thực thi trên nhiều ngôn ngữ với nhiều thể hiện của trạng thái bên trong, và do đó nó có thể cung cấp con trỏ đến các hàm thành viên của giao diện.

3- Giao diện có kiểu mạnh: Mỗi một giao diện đều có một định danh riêng nên ngăn chặn được khả năng xung đột có thể xảy ra đối với các tên được đặt cho giao diện. Điều này tăng thêm tính bền vững cho chương trình.

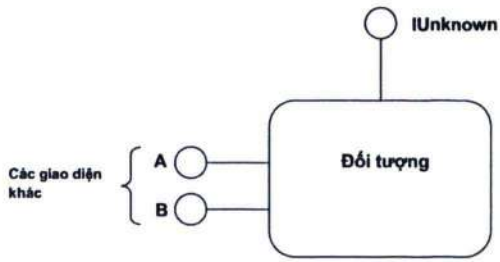
4- Các giao diện được phân biệt rõ ràng: Mọi sự thay đổi như thêm hoặc xoá hàm thành viên, thay đổi ngữ nghĩa đều dẫn tới hình thành một giao diện mới và được gán một định danh mới. Do đó giao diện mới và cũ không thể xung đột với nhau cho dù mọi sự thay đổi chỉ đơn thuần là về ngữ nghĩa.

6.2.4.4.2. Đối tượng COM

Một đối tượng COM có thể được lập trình bằng một ngôn ngữ thông dụng như C, C++ hoặc VB. Một đối tượng có thể cung cấp nhiều giao diện. Tất cả các đối tượng COM đều có một giao diện cơ bản là IUnknown. Đây là giao diện cơ sở cho tất cả các giao diện khác trong COM mà mọi đối tượng phải hỗ trợ. Bên cạnh đó, đối tượng cũng có khả năng thực thi nhiều giao diện khác.

Các đối tượng với nhiều giao diện có thể cung cấp các con trỏ truy nhập vào nhiều bảng chứa các hàm. Các con trỏ này có thể gọi được con trỏ giao diện.

Trong COM, giao diện là một bảng các con trỏ (giống như table trong C++) vào các hàm được thực hiện bởi đối tượng.

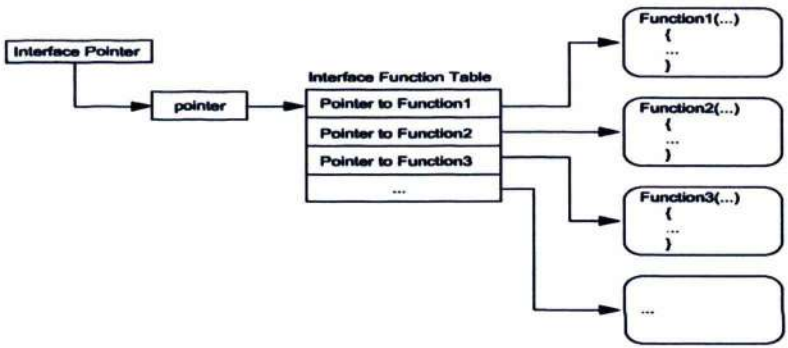


Hình 6.20. Mô hình một đối tượng COM [26]

Trong thực tế, một con trỏ trỏ đến một giao diện là một con trỏ tới một con trỏ trỏ tới bảng các con trỏ vào các hàm thành viên. Tuy nhiên, để tránh cách diễn đạt quanh co này khi nói về giao diện người ta thường sử dụng thuật ngữ con trỏ giao diện để thay thế. Khi đó sự thực thi giao diện đơn giản là dùng con trỏ trỏ tới mảng các con trỏ tới các hàm.

Như đã trình bày ở trên, mọi đối tượng COM đều hỗ trợ giao diện IUnknown. Giao diện này có ba phương thức AddRef(), Release() và QueryInterface(). Tất cả các giao diện khác đều dẫn xuất từ giao diện IUnknown và đều có các con trỏ đến các phương thức này.

Hai phương thức đầu tính toán số đếm tham chiếu để điều khiển thời gian sống của đối tượng. Khi đối tượng được tạo lần đầu, ta cần gọi phương thức AddRef() của đối tượng để tăng số đếm. Khi không còn cần tới đối tượng, ta gọi phương thức Release() để giảm số đếm tham chiếu. Khi người dùng cuối cùng gọi phương thức Release(), số đếm giảm về 0 thì đối tượng sẽ tự huỷ.



Hình 6.21. Sự thực thi con trỏ giao diện [26]

Khi ta có một con trỏ đến đối tượng thì thực chất, những gì có được chỉ là một con trỏ đến một trong số các giao diện của nó, còn đó là giao diện nào thì lại phụ thuộc vào cách mà chúng ta có con trỏ đó. Từ con trỏ vào một giao diện, có thể truy cập được con trỏ vào các giao diện khác mà đối tượng hỗ trợ. Đối tượng có thể hoặc không hỗ trợ giao diện, nhưng mọi đối tượng đều được bảo đảm hỗ trợ giao diện Iunknown nên có thể yêu cầu các giao diện khác qua phương thức:

IUnknow::QueryInterface().

Mối quan hệ giữa đối tượng và các giao diện thể hiện ở chỗ:

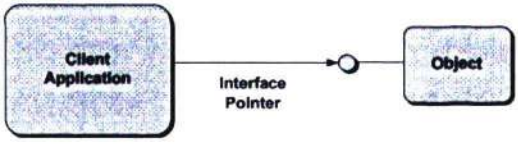
- Các Client chỉ kết nối qua con trỏ tới các giao diện. Khi một Client truy nhập vào một đối tượng, Client chỉ đơn thuần thông qua con trỏ giao diện. Con trỏ che đi nội dung của thao tác bên trong, không thể thấy chi tiết về đối tượng mà chỉ có thể thấy thông tin về trạng thái của chúng.

- Đối tượng có thể thực thi nhiều giao diện. Một lớp thực thi đối tượng có thể thực thi nhiều giao diện, ví dụ qua phương pháp đa thừa kế.

6.2.4.4.3. Giao tiếp giữa Client và Object

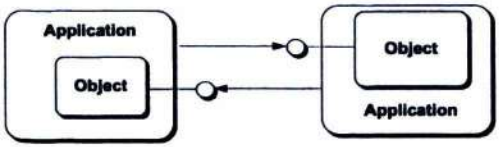
a. Cách thức của sự giao tiếp

Trước khi sử dụng một đối tượng COM trong một ứng dụng, ta cần khởi tạo cơ chế COM trong ứng dụng bằng lời gọi CoInitialize(...) và sau đó tạo đối tượng COM mong muốn. Client kết nối với Object thông qua con trỏ giao diện và không bao giờ truy nhập trực tiếp vào Object. Khi cần sử dụng dịch vụ nào đó của đối tượng, Client hiểu rằng nó cần có con trỏ đến một hay nhiều giao diện của đối tượng. Để tạo một đối tượng COM và nhận một con trỏ vào giao diện, ta có thể gọi một trong hai hàm CoCreateInstance() hoặc CoCreateInstanceEx() với các tham số xác định đối tượng.



Hình 6.22. Giao tiếp giữa đối tượng và khách hàng [26]

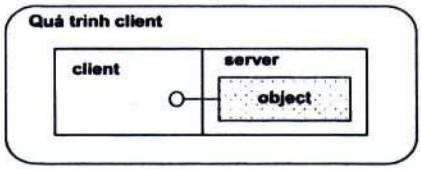
Trong một số trường hợp, bản thân Client sẽ đóng vai trò một object và cung cấp cho các đối tượng khác những chức năng gọi các sự kiện hoặc đưa ra các dịch vụ của nó. Lúc này Client là một đối tượng thực thi còn Object là một khách hàng.



Hình 6.23. Giao tiếp giữa hai đối tượng [26]

b. Giao tiếp trên cùng một quá trình tính toán

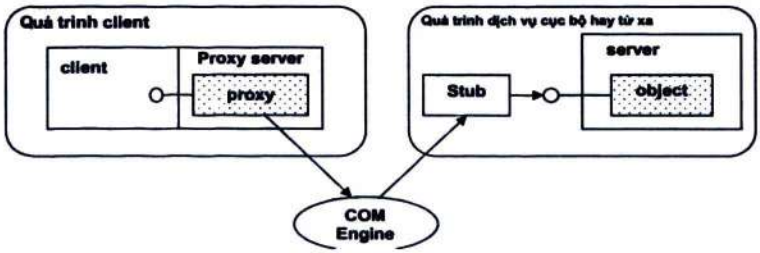
Khi Client và đối tượng COM cùng nằm trên một quá trình tính toán thì Client sẽ kết nối trực tiếp với Object qua con trỏ giao diện.



Hình 6.24. Giao tiếp giữa đối tượng và khách hàng trên cùng quá trình [26]

c. Giao tiếp liên quá trình

Nếu Client và Object không cùng nằm trên một không gian địa chỉ hay nằm trên các máy tính khác nhau thì COM sẽ thiết lập một đối tượng đại diện (proxy) bên phía Client và một đối tượng gốc (stub) bên phía object. Proxy và stub sẽ kết nối với nhau qua kênh giao tiếp (channel). Khi đó, Client sẽ thực hiện lời gọi dịch vụ trong không gian địa chỉ của nó tức là giao tiếp trực tiếp với proxy. Proxy sẽ thu thập (marshal) các thông số, gửi chúng đến stub qua kênh giao tiếp. Stub thực hiện lời gọi đến đối tượng dịch vụ, đóng gói kết quả và đưa về cho proxy.



Hình 6.25. Giao tiếp giữa đối tượng và khách hàng trên hai quá trình khác nhau [26]

6.2.4.4.4. Ngôn ngữ mô tả giao diện

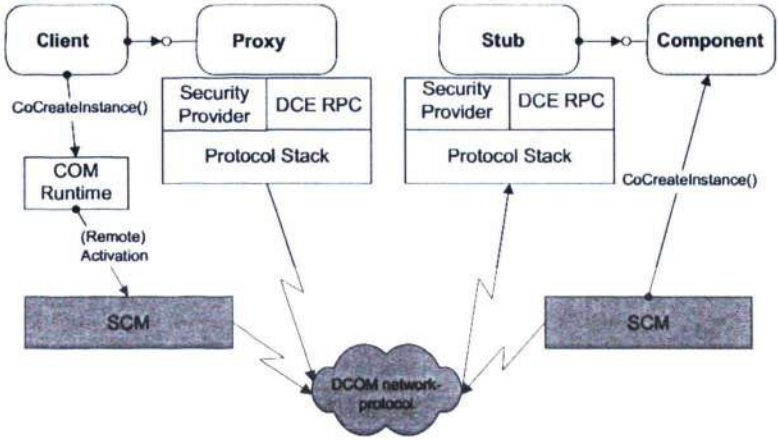
IDL (Interface Description Language) là một ngôn ngữ kiểu mạnh dùng để mô tả giao diện của đối tượng COM, độc lập với ngôn ngữ lập trình. Cú pháp của ngôn ngữ này không phức tạp so với một ngôn ngữ lập trình. Khi xây dựng một đối tượng COM, cần mô tả các phương thức của giao diện bằng cách sử dụng ngôn ngữ này. Sau khi nạp xong file mô tả giao diện này, cần lưu nó ở dạng *.idl để chương trình dịch có thể hiểu được. Chương trình dịch (IDL-Compiler) sẽ dịch sang một ngôn ngữ lập trình, ví dụ C++. Khi đó một giao diện sẽ được chuyển sang thực hiện bằng một cấu trúc thích hợp trong ngôn ngữ lập trình, ví dụ một lớp thuần ảo trong C++.

6.2.4.4.5. Mô hình đối tượng thành phần phân tán DCOM

DCOM (Distributed COM) mở rộng COM cho việc giao tiếp giữa các đối tượng phân tán, thuộc các chương trình chạy trên nhiều máy tính khác nhau trên mạng LAN, WAN hay Internet. Với DCOM, các ứng dụng có thể phân tán trên nhiều vị trí đem lại sự thuận lợi cho chính ứng dụng. Ngày nay khi người ta nói tới COM là cũng thường bao hàm DCOM trong đó.

DCOM là một công nghệ lý tưởng cho những ứng dụng nhiều tầng lớp bởi vì nó cho phép những thành phần ActiveX làm việc ngang qua mạng. Nhiều người có thể phát triển thêm cùng một thành phần mà không cần phải lo lắng về lập trình mạng, tính tương thích hệ thống hoặc sự hợp nhất của những thành phần xây dựng từ những ngôn ngữ khác nhau. Nó dẫn tới hạ thấp giá thành và làm giảm sự phức tạp của việc phân tán các ứng dụng thành phần.

Khi các đối tượng ở trên các máy tính khác nhau, DCOM đơn giản thực hiện sự thay thế truyền thông liên quá trình cục bộ bởi giao thức mạng. Hình dưới đây minh họa rõ nét cách thức giao tiếp giữa các đối tượng nằm trên hai máy tính khác nhau.



Hình 6.26. Giao tiếp giữa đối tượng và khách hàng trên hai máy khác nhau với DCOM [26]

Thư viện COM Run-Time cung cấp những dịch vụ hướng đối tượng tới khách hàng và thành phần muốn giao tiếp với nhau đồng thời sử dụng RPC và nhà cung cấp an toàn để tạo chuẩn nối mạng đóng gói tuân theo giao thức truyền thông cho DCOM.

6.3. Đặc tả kiến trúc hệ thống được thiết kế theo hướng dịch vụ

Trong chương 1, phần tổng quan về kỹ nghệ hướng dịch vụ đã được giới thiệu. Trong phần này, chúng ta tập trung chỉ vào đặc tả kiến trúc hệ thống được thiết kế theo hướng dịch vụ.

6.3.1. Kiểu kiến trúc hệ thống được thiết kế theo hướng dịch vụ

SOA là một kiểu kiến trúc với mục tiêu kết nối lỏng lẻo các thành phần phần mềm có thể tương tác với nhau. Đó là một kiểu kiến trúc *công nghệ thông tin (Information Technology- IT)* tập trung cho doanh nghiệp nhằm hỗ trợ việc tích hợp nghiệp vụ như các nhiệm vụ nghiệp vụ có kết nối nhau và được thực hiện lặp đi lặp lại, hoặc là các dịch vụ. Còn dịch vụ là một đơn vị công việc do một nhà cung cấp dịch vụ thực hiện nhằm để đạt kết quả kỳ vọng cuối cùng cho một người tiêu dùng dịch vụ.

Enterprise Architecture, 2008, California Franchise Tax Board đưa ra một định nghĩa về SOA: SOA là một cách tiếp cận kiến trúc kiểu IT hướng trọng tâm vào kinh doanh nhằm tích hợp các tiến trình kinh doanh thông qua việc thực thi các nhiệm vụ có tính chất lặp đi lặp lại. Trong đó, SOA Registry và Repository là một trung tâm tham chiếu đến tất cả các thành phần phần mềm trong môi trường SOA.

Dịch vụ được hiểu là một tập rời rạc các chức năng nghiệp vụ hoặc kỹ thuật xác định. Với tập đầu vào và đầu ra xác định, các dịch vụ này có thể được dùng lại. Tất cả những gì diễn ra đằng sau giao diện là được che dấu và không có liên quan đến người tiêu dùng dịch vụ.

Các công nghệ của các dịch vụ Web là công nghệ kết nối thích hợp nhất của SOA. Các dịch vụ Web về cơ bản sử dụng XML để tạo ra một kết nối thiết thực.

Kiến trúc SOA không hoàn toàn mới, DCOM và CORBA cũng có kiến trúc tương tự. Tuy nhiên, *các kiến trúc cũ ràng buộc các thành phần với nhau quá chặt*, ví dụ như các ứng dụng phân tán muốn làm việc với nhau phải đạt được "thỏa thuận" về chi tiết tập hàm API, một thay đổi mã lệnh trong thành phần COM sẽ yêu cầu những thay đổi tương ứng đối với mã lệnh truy cập thành phần COM này.

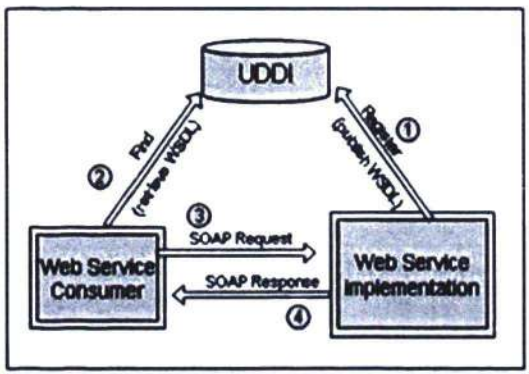
SOA là kiến trúc phần mềm bắt nguồn từ định nghĩa giao tiếp và xây dựng toàn bộ mô hình ứng dụng như là mô hình các giao tiếp, hiện thực giao tiếp và phương thức gọi giao tiếp. *Giao tiếp là trung tâm của toàn bộ triết lý kiến trúc này*. Dịch vụ và mô-đun phần mềm nghiệp vụ được truy cập thông qua giao tiếp, thường theo cách thức yêu cầu - phản hồi. Ngay cả với yêu cầu dịch vụ một chiều thì nó vẫn là yêu cầu trực tiếp có chủ đích từ một phần mềm này đến một phần mềm khác. Một tương tác hướng dịch vụ luôn bao hàm một cặp đối tác: nguồn cung cấp dịch vụ và khách hàng sử dụng dịch vụ.

6.3.2. Mô hình kiến trúc hướng dịch vụ

Với mục tiêu là xây dựng một kiến trúc dịch vụ thực sự đơn giản và tính tương thích cao, SOA được xây dựng dựa trên các chuẩn rất phổ biến là SOAP (Simple Object Access Protocol -giao thức truy xuất đối tượng đơn giản) và

XML (eXtensible Makup Language -ngôn ngữ đánh dấu mở rộng). Hai chuẩn này đóng vai trò là thành phần xương sống để truyền nhận các thông điệp giữa các đối tượng truyền thông. Tức là, bất kỳ thành phần nào nếu hiểu được giao thức này thì hoàn toàn có thể sử dụng các dịch vụ mà không phụ thuộc vào ngôn ngữ lập trình hay hệ điều hành đang sử dụng.

Mô hình của kiến trúc có thể mô tả như sau:



Hình 6.27. Mô hình kiến trúc hướng dịch vụ [28]

Các thực thể trong kiến trúc hướng dịch vụ bao gồm:

1. Dịch vụ (Services);
2. Thành phần sử dụng dịch vụ (Services consumer / Client / Request);
3. Thành phần cung cấp dịch vụ (Services Provider);
4. Thành phần đăng ký dịch vụ (Services Registry);
5. Giao ước dịch vụ (Contract);
6. Uỷ nhiệm dịch vụ;
7. Ràng buộc sử dụng dụng dịch vụ (Services lease).

6.3.2.1. Dịch vụ

Dịch vụ chứa một chức năng rõ ràng, tự chứa đựng và không phụ thuộc vào ngữ cảnh hay trạng thái của các dịch vụ khác. Các thành phần sử dụng dịch vụ có thể truy cập tới dịch vụ thông qua giao diện dịch vụ được xuất bản. Dịch vụ có các tính chất sau:

1- Dịch vụ có tính chất rõ ràng, là một đơn vị chức năng nghiệp vụ có thể được gọi;

2- Có khả năng gọi dịch vụ thông qua các giao thức truyền thông chung;

3- Dịch vụ có tính liên thông và vị trí trong suốt;

4- Dịch vụ được định nghĩa bằng các giao diện tường minh;

5- Các giao diện của dịch vụ độc lập với cài đặt;

6- Dịch vụ cung cấp giao ước giữa các thành phần cung cấp và sử dụng dịch vụ;

7- Dịch vụ là các mô-đun phức tạp, bao gồm nhiều thành phần. Mức độ đóng gói của dịch vụ càng cao thì dịch vụ càng có khả năng tái sử dụng và linh hoạt.

6.3.2.2. Thành phần sử dụng dịch vụ (Web Services Customer)

Thành phần sử dụng dịch vụ là một ứng dụng, một dịch vụ, hoặc một loại mô-đun phần mềm khác có yêu cầu sử dụng dịch vụ. Đây là thực thể khởi tạo việc định vị tại một kho đăng ký dịch vụ, liên kết tới dịch vụ qua một kênh truyền thông và thực thi các chức năng của dịch vụ. Thành phần này thực thi nhiệm vụ bằng cách gửi tới dịch vụ một yêu cầu được định dạng theo đúng giao ước.

6.3.2.3. Thành phần cung cấp dịch vụ (Web Services Implementation)

Thành phần cung cấp dịch vụ là một thực thể có khả năng được địa chỉ hoá qua mạng, nó có thể chấp nhận và thực thi các yêu cầu từ những thành phần sử dụng dịch vụ. Thành phần cung cấp dịch vụ có thể là một hệ thống máy tính lớn, một thành phần, hoặc một loại hệ thống phần mềm khác có thể thực thi các yêu cầu dịch vụ. Thực thể này xuất bản giao ước dịch vụ của nó trong một kho đăng ký dịch vụ để các thành phần sử dụng dịch vụ có thể truy cập.

6.3.2.4. Thành phần đăng ký dịch vụ

Thành phần đăng ký dịch vụ là một thư mục trên mạng có chứa các dịch vụ sẵn dùng. Đây là một thực thể chấp nhận và lưu trữ các giao ước từ các thành phần cung cấp dịch vụ và cung cấp các giao ước đó cho các thành phần sử dụng dịch vụ.

UDDI (Universal Description, Discovery and Integration) là thành phần đăng ký dịch vụ, đóng vai trò như người chỉ đường cho các thành phần sử dụng dịch vụ, tạo sự trong suốt về vị trí đối với thành phần cung cấp dịch vụ; để khi thành phần này thay đổi thì thành phần sử dụng dịch vụ không cần phải biên dịch hay cấu hình lại.

Giữa ba thành phần: cung cấp dịch vụ, sử dụng dịch vụ, đăng ký dịch vụ thì việc trao đổi dữ liệu sẽ hoàn toàn dùng giao thức SOAP và nội dung bên trong được định dạng bằng XML.

6.3.2.5. Giao ước dịch vụ

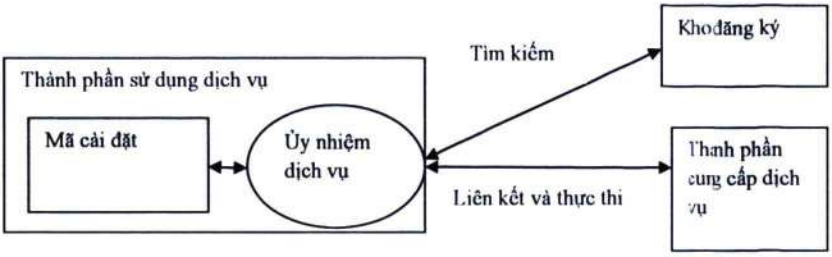
Một giao ước là một bản đặc tả cách thức để thành phần sử dụng dịch vụ có thể tương tác với thành phần cung cấp dịch vụ. Nó chỉ ra khuôn dạng của thông điệp yêu cầu và thông điệp đáp ứng từ các dịch vụ. Giao ước dịch vụ có thể đòi hỏi một tập các điều kiện tiên quyết và điều kiện sau. Các điều kiện này xác định trạng thái cần thiết của dịch vụ để thực thi một chức năng cụ thể. Bản giao ước này cũng có thể bao gồm các mức độ chất lượng của dịch vụ, các đặc tả cho các yếu tố phi chức năng của dịch vụ.

6.3.2.6. Uỷ nhiệm dịch vụ

Thành phần cung cấp dịch vụ cung cấp một uỷ nhiệm dịch vụ cho thành phần sử dụng dịch vụ. Thành phần sử dụng dịch vụ thực thi các yêu cầu bằng cách gọi một hàm API trên nó. Uỷ nhiệm dịch vụ (hình 6.27) sẽ tìm một giao ước và một tham chiếu tới thành phần cung cấp dịch vụ ở trụ sở đăng ký. Sau đó, nó định dạng thông điệp yêu cầu và thực thi yêu cầu trên danh nghĩa của thành phần sử dụng dịch vụ. Uỷ nhiệm dịch vụ là một thực thể không bắt buộc, nó chỉ đơn giản hoá cho thành phần sử dụng dịch vụ và thành phần sử dụng dịch vụ hoàn toàn có thể viết phần mềm để truy cập tới dịch vụ.

Một thành phần cung cấp dịch vụ sẽ cung cấp nhiều uỷ nhiệm cho các môi trường khác nhau, mỗi uỷ nhiệm dịch vụ được viết bằng ngôn ngữ của các thành phần sử dụng dịch vụ. Ví dụ, một thành phần cung cấp dịch vụ có thể cung cấp các uỷ nhiệm dịch vụ cho Java, Visual Basic, Delphi nếu đó là các nền tảng của các thành phần sử dụng dịch vụ. Mặc dù uỷ nhiệm dịch vụ là

không bắt buộc nhưng có thể cải thiện một cách đáng kể hiệu năng và tính tiện dụng cho các thành phần sử dụng dịch vụ.



Hình 6.28. Uỷ nhiệm dịch vụ [28]

6.3.2.7. Ràng buộc sử dụng dịch vụ

Ràng buộc sử dụng dịch vụ mà các thành phần đăng ký dịch vụ gán cho thành phần sử dụng dịch vụ rất cần thiết để dịch vụ bảo trì được thông tin trạng thái liên kết giữa thành phần sử dụng và thành phần cung cấp. Nó tạo ra sự gắn kết không chặt chẽ giữa các thành phần này bằng cách giới hạn khoảng thời gian mà chúng được liên kết với nhau. Không ràng buộc, một thành phần sử dụng dịch vụ có thể liên kết với một dịch vụ mãi mãi và không bao giờ liên kết lại với các giao ước của nó.

6.3.3. Các nguyên tắc thiết kế kiến trúc hướng dịch vụ

Kiến trúc hướng dịch vụ dựa trên 2 nguyên tắc thiết kế rất quan trọng:

1-Mô-đun: Tách vấn đề lớn thành nhiều vấn đề nhỏ;

2-Đóng gói: Che đi dữ liệu và logic trong từng mô-đun (hay "hộp đen") đối với truy cập từ ngoài.

Có ý kiến cho rằng SOA sẽ thay thế phương thức lập trình hướng đối tượng - OOP (Object Oriented Programming).

SOA sử dụng cùng số nguyên lý như OOP, tuy nhiên triết lý SOA có khác biệt đáng kể so với OOP. SOA có thể một thực hiện với cả chương trình theo *hướng đối tượng (OO)* và chương trình không hướng đối tượng

Hình 6.29. Kiến trúc 3 tầng tiêu biểu của mô hình đối tượng [9]

Mô hình đối tượng có *kiến trúc 3 tầng (three-ties):*

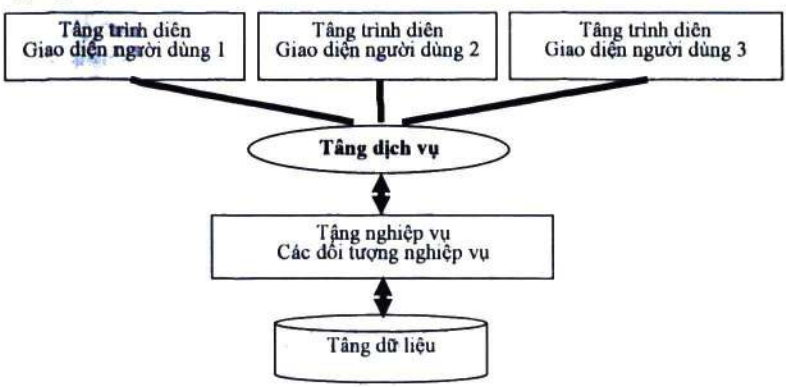
1-Tầng thể hiện;

2-Tầng nghiệp vụ;

3-Tầng cơ sở dữ liệu.

Có sự ràng buộc giữa tầng thể hiện và các đối tượng nghiệp vụ của tầng nghiệp vụ. Chương trình người dùng phải tương tác với mô hình đối tượng của tầng nghiệp vụ, điều này làm tăng sự ràng buộc và số lượng đáng kể các "gọi hàm" giữa 2 tầng. Khi các đối tượng nghiệp vụ nằm ở máy tính xa thì đây sẽ là vấn đề.

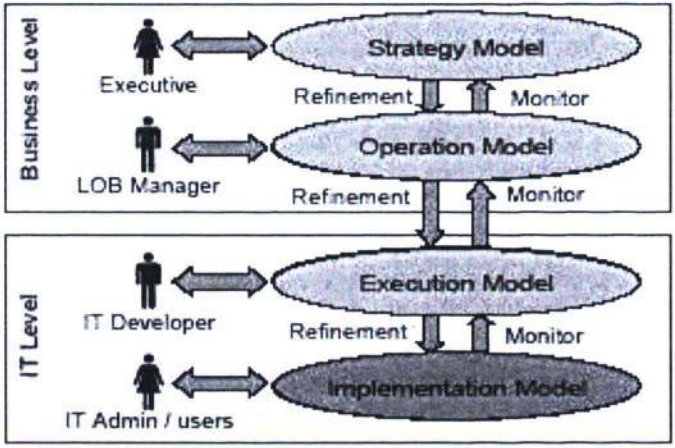
Tương tự, số lượng đối tượng nghiệp vụ mà tầng thể hiện phải thao tác làm giảm sự độc lập giữa các tầng và gây khó khăn cho việc sử dụng tầng nghiệp vụ.



Hình 6.30. Mô hình SOA phát triển lên từ mô hình đối tượng bằng cách thêm tầng dịch vụ [9]

Khác biệt giữa SOA và mô hình đối tượng chính là có thêm tầng mới "Services" nằm giữa tầng thể hiện và tầng nghiệp vụ. Tầng thể hiện giờ đây không còn thao tác trực tiếp lên các đối tượng nghiệp vụ nữa, mà sử dụng dịch vụ để truy cập chúng. Các đối tượng nghiệp vụ được đặt trong thư viện và được dịch vụ nạp vào bộ nhớ - tầng dịch vụ và tầng nghiệp vụ nằm trong cùng tiến trình, nhờ vậy lời gọi hàm đến đối tượng nghiệp vụ sẽ không bị quá tải.

Dịch vụ đóng vai trò như "hộp đen", cung cấp một lớp trung gian cho mô hình đối tượng và đưa ra tập chức năng rút gọn, làm giảm nhu cầu trao đổi giữa các tầng.



Hình 6.31. Mô hình kiến trúc SOA của IBM [14]

Hình 6.31 mô tả một khung công việc theo kiến trúc hướng dịch vụ được đề xuất bởi IBM. Hai mức kinh doanh nghiệp vụ và CNTT được định nghĩa. Mỗi mức có hai mô hình. Mức kinh doanh nghiệp vụ có mô hình chiến lược và tác nghiệp; mức CNTT có mô hình thực hiện và triển khai. Mỗi mô hình được xác định rõ ràng những ai có liên quan với mô hình [9].

TÓM TẮT CHƯƠNG

Các hệ thống phân tán được thiết kế và phát triển. Chúng có một ý nghĩa thực tiễn to lớn đối với các hoạt động hàng ngày của tổ chức và là một thành tựu mới của việc phát triển một HTTT.

Khi thiết kế hệ thống phân tán cần chú ý đến tính cân đối giữa ba yếu tố có thể tác động đến nó là tính khả thi, sự sống và tính sẵn sàng của hệ thống. Để có được thiết kế hiệu quả, cần nắm vững những đặc trưng của kiến trúc được sử dụng để trợ giúp các hệ thống phân tán và chia sẻ dữ liệu.

Để đáp ứng được mục tiêu cung cấp dịch vụ truy nhập dữ liệu cho NSD ở mọi nơi khác nhau, hệ thống phân tán cần có tính "*trong suốt địa phương*".

Các ứng dụng xây dựng trên cơ sở kiến trúc Client-Server cũng khác với hệ thống CSDL tập trung trên một máy lớn. Điều khác cơ bản đó là, mỗi máy khách là một phần thông minh của hệ thống xử lý ứng dụng.

Hiện nay, người ta đã đưa vào mô hình Client-Server nhiều chức năng hệ thống ứng dụng khác nhau dựa trên ba thành phần là 1-Quản trị dữ liệu, 2-Trình diễn dữ liệu, 3-Xử lý dữ liệu. Người ta cũng đã phát triển nhiều dạng tiên tiến phổ dụng của kiến trúc Client-Server.

Trong UML, kiến trúc của hệ thống phần mềm chuyên sâu được mô tả theo 5 loại khung nhìn (views) khác nhau. Mỗi khung nhìn phản ánh về một khía cạnh của tổ chức và cấu trúc của hệ thống, tập trung vào từng mặt cụ thể giúp cho ta hiểu và sử dụng hệ thống tốt nhất. Mỗi khung nhìn thường được thể hiện trong một số sơ đồ nhất định.

Phân tích kiến trúc theo hướng đối tượng bao gồm việc chính là *xác định các gói, các lớp thực thể hiển nhiên*. Thiết kế kiến trúc theo hướng đối tượng bao gồm việc thiết kế các cấu hình mạng, thiết kế các hệ thống con và các giao diện của chúng, xác định các lớp quan trọng về mặt kiến trúc.

Đối tượng phân tán cũng là một hướng phát triển tự nhiên từ phương pháp luận hướng đối tượng. Trong khi kỹ nghệ hướng thành phần quan tâm tới việc đóng gói các đối tượng để có thể sử dụng lại một cách thuận tiện, thì kỹ

nghệ đối tượng phân tán tập trung vào vấn đề kiến trúc các đối tượng có khả năng giao tiếp một cách trong suốt trên các nền và hệ thống mạng khác nhau

Một đối tượng phân tán là một đối tượng phần mềm trong một hệ thống phân tán, có thể được sử dụng bởi các chương trình ứng dụng hoặc các đối tượng khác thuộc cùng một quá trình tính toán, thuộc một quá trình tính toán khác hoặc thuộc một trạm khác trong mạng theo một phương thức thống nhất, không để ý tới giao thức truyền thông cụ thể, trong suốt với hệ điều hành, kiến trúc phần cứng và hệ thống mạng.

Một đối tượng phân tán có các thuộc tính có thể truy cập được từ xa, có các phép toán có thể gọi được từ xa.

Một kiến trúc đối tượng phân tán định nghĩa mô hình các đối tượng phân tán, mô hình giao tiếp và chuẩn giao tiếp giữa các đối tượng phân tán.

Một kiến trúc đối tượng phân tán tạo điều kiện cho việc lập trình ở một mức trừu tượng cao hơn so với phương pháp hướng đối tượng cổ điển. Điểm khác biệt so với lập trình hướng đối tượng cổ điển nằm ở "*tính trong suốt phân tán*".

SOA là kiến trúc phần mềm bắt nguồn từ định nghĩa giao tiếp và xây dựng toàn bộ mô hình ứng dụng như là mô hình các giao tiếp, hiện thực giao tiếp và phương thức gọi giao tiếp. *Giao tiếp là trung tâm của toàn bộ triết lý kiến trúc này.*

SOA được xây dựng dựa trên các chuẩn rất phổ biến là SOAP và XML. Hai chuẩn này đóng vai trò là thành phần xương sống để truyền nhận các thông điệp giữa các đối tượng truyền thông.

Kiến trúc hướng dịch vụ được thiết kế dựa trên 2 nguyên tắc rất quan trọng:

1-Mô-đun: Tách vấn đề lớn thành nhiều vấn đề nhỏ;

2-Đóng gói: Che đi dữ liệu và logic trong từng mô-dun đối với truy cập từ ngoài.

CÂU HỎI CHƯƠNG 6

6.1. Khi thiết kế hệ thống phân tán, người ta quan tâm đến những khía cạnh nào?

6.2. Kiến trúc Client-Server có các ưu điểm gì?

6.3. Hiện nay, người ta đã đưa vào mô hình Client-Server nhiều chức năng hệ thống ứng dụng khác nhau dựa trên ba thành phần nào?

6.4. Tại sao nói "lấy kiến trúc làm trung tâm" là đặc trưng nổi bật của quá trình phát triển?

6.5. Phân tích kiến trúc bao gồm những việc chính nào?

6.6. Thiết kế kiến trúc bao gồm những thiết kế con nào?

6.7. Một đối tượng phân tán là gì?

6.8. Tính *"trong suốt phân tán"* được thể hiện qua các đặc tính nào?

6.9. Khái niệm "dịch vụ" trong SOA được hiểu như thế nào?

6.10. SOA được xây dựng dựa trên các chuẩn nào?. Ý nghĩa, vai trò của các chuẩn đó?

BÀI TẬP CHƯƠNG 6

Bài tập 6.1

Hệ thống rút tiền tự động ATM được mô tả trong bài tập 3.2 chương 3 có sự tham gia của các đối tượng thuộc các lớp: MayDocThe, ManHinh, TaiKhoan và MayTraTien. Yêu cầu vẽ biểu đồ trình tự của ATM mô tả ca sử dụng "Rút tiền"*?*

Bài tập 6.2

Hệ thống rút tiền tự động ATM được mô tả trong bài tập 3.2 chương 3 có sự tham gia của các đối tượng thuộc các lớp: **MayDocThe, ManHinh, TaiKhoan** và **MayTraTien.** Yêu cầu vẽ *biểu đồ cộng tác mô tả ca sử dụng "Rút tiền"*?

Bài tập 6.3

Hệ thống rút tiền tự động ATM được mô tả trong bài tập 3.2 chương 3 có sự tham gia của các đối tượng thuộc các lớp: **MayDocThe, ManHinh, TaiKhoan, MayTraTien và HeThongTinDung.** Yêu cầu vẽ *biểu đồ lớp chi tiết*?

Chương 7
BẢO ĐẢM CHẤT LƯỢNG PHẦN MỀM

Phần mềm chất lượng cao là mục tiêu quan trọng trong quá trình phát triển phần mềm.

Chương này trình bày các vấn đề về bảo đảm chất lượng phần mềm.

Học viên cần nắm vững khái niệm bảo đảm chất lượng sản phẩm nói chung và sản phẩm phần mềm nói riêng, hiểu rõ các chuẩn đánh giá chất lượng sản phẩm phần mềm và các mô hình chất lượng, việc đo chất lượng sản phẩm phần mềm, bao quát được các hoạt động bảo đảm chất lượng cũng như tiến trình triển khai bảo đảm chất lượng.

7.1. Khái niệm bảo đảm chất lượng

Theo định nghĩa hình thức về chất lượng phần mềm của Tổ chức tiêu chuẩn quốc tế ISO trong bộ tiêu chuẩn 8402: "*Chất lượng là khả năng đáp ứng toàn diện nhu cầu của người dùng về tính năng cũng như cũng như công dụng được nêu ra một cách tường minh hoặc không tường minh trong những ngữ cảnh xác định*".

Theo Roger S. Pressman [23], chất lượng phần mềm được xác định bằng "*việc tuân thủ các yêu cầu chức năng và sự hoàn thiện đã được phát biểu tường minh, các chuẩn phát triển đã được tư liệu hóa tường minh, và các đặc trưng không tường minh được trông đợi từ tất cả các phần mềm đã được phát triển theo cách chuyên nghiệp*".

Những năm cuối thế kỷ 20, tổ chức ISO đã tập trung rất nhiều vào các tiêu chuẩn chất lượng cho phần mềm. Theo cách tiếp cận của ISO, chất lượng toàn diện của phần mềm cần được quan tâm từ chất lượng quy trình, chất lượng của nội tại sản phẩm phần mềm, chất lượng đáp ứng yêu cầu người dùng cũng

như chất lượng sử dụng phần mềm. Đó chính là mô hình chất lượng ISO-9126 được xây dựng dựa trên cơ sở phân loại và chia nhỏ những thuộc tính chất lượng, nhằm tạo được những đại lượng đo đếm được dùng để kiểm định chất lượng của sản phẩm phần mềm.

Chất lượng phần mềm được đánh giá trước hết bởi một số đặc tính. Các đặc tính này có thể được chia thành các đặc tính *chất lượng trong* và *chất lượng ngoài*. Chất lượng ngoài là những gì lộ ra khi một người dùng vận hành phần mềm. Chất lượng trong đề cập đến các khía cạnh phụ thuộc vào mã, không hiển thị rõ cho người dùng biết. Chất lượng ngoài là rất quan trọng cho người dùng, trong khi chất lượng trong lại rất có ý nghĩa cho nhà phát triển.

Bảo đảm chất lượng phần mềm (*Software Quality Assurance*-SQA) là một hoạt động được áp dụng trong toàn bộ tiến trình kỹ nghệ phần mềm, bao phủ các quá trình phân tích, thiết kế, mã hóa, kiểm thử, các cuộc họp chính thức, các chiến lược kiểm thử, quá trình kiểm soát tư liệu, sự tuân thủ các chuẩn, cơ chế đo đạc và báo cáo.

Bảo đảm chất lượng phần mềm bao gồm một số phương tiện để giám sát các quy trình công nghệ phần mềm mà các phương pháp được sử dụng để bảo đảm chất lượng. Các phương pháp này rất nhiều và đa dạng, có thể bao gồm bảo đảm sự phù hợp với một hoặc nhiều tiêu chuẩn, chẳng hạn như ISO 9000 hoặc mô hình *chỉ số trưởng thành phần mềm (Software Maturity Index- SMI)*.

Bảo đảm chất lượng phần mềm là những cách thức và hành động để chắc chắn rằng cơ chế kiểm soát chất lượng phần mềm là thích hợp và đang được áp dụng chính xác, do vậy đem lại niềm tin cho khách hàng.

7.2. Mô hình chất lượng

7.2.1. Các chuẩn đánh giá chất lượng sản phẩm phần mềm

7.2.1.1. Chuẩn ISO

Bằng việc bổ sung vào một hệ thống quản lý chất lượng chung, các chuẩn của ISO về phần mềm giúp chúng ta quản lý chất lượng theo cách chuyên nghiệp nhất [12].

Các chuẩn phần mềm là rất quan trọng vì những lý do sau:

- Chúng dựa trên hiểu biết về những thực tiễn thích hợp nhất cho công ty. Kinh nghiệm này thường chỉ đạt được sau rất nhiều lần thử nghiệm và lỗi. Bổ sung nó vào các chuẩn giúp cho công ty tránh lặp lại sai lầm trong quá khứ.

- Chúng cung cấp một khung cho việc thực thi quá trình bảo đảm chất lượng, đưa ra các chuẩn mực thực tiễn.

- Chúng trợ giúp tính liên tục khi mà một người tiếp tục công việc của người khác bỏ dở. Do vậy, công sức nghiên cứu mới được giảm xuống.

ISO 9000 là một hệ thống quản lý chất lượng chung. Hệ thống này bảo đảm rằng về mặt hệ thống hoá thì các thủ tục sẽ được đưa ra, các kết quả sẽ được *tài liệu hoá*.

Các chuẩn tài liệu trong một dự án phần mềm là quan trọng bởi vì các tài liệu là minh chứng để thể hiện phần mềm và quá trình phần mềm.

Có ba kiểu chuẩn tài liệu:

1- *Các chuẩn quá trình tài liệu*: Những chuẩn này định ra quá trình mà sẽ được tuân theo đối với việc tạo ra tài liệu.

2- *Các chuẩn tài liệu*: Những chuẩn này chi phối cấu trúc và cách thể hiện của các tài liệu.

3- *Các chuẩn trao đổi tài liệu*: Những chuẩn này bảo đảm rằng tất cả các bản sao điện tử của các tài liệu là tương thích.

Các chuẩn tài liệu phải được áp dụng cho tất cả các tài liệu được tạo ra trong một dự án phát triển phần mềm

Các chuẩn tài liệu có thể được phát triển bao gồm:

1- *Các chuẩn nhận dạng tài liệu*: với hàng nghìn tài liệu, mỗi tài liệu phải được nhận biết duy nhất. Đối với các tài liệu chính quy, nhận dạng này có thể là nhận dạng chính quy được người quản lý cấu hình định ra. Đối với các tài liệu không chính quy, người quản lý dự án có thể định ra mẫu của tài liệu.

2- *Các chuẩn cấu trúc tài liệu*: Mỗi lớp của tài liệu được tạo ra trong dự án phần mềm phải theo một số cấu trúc chuẩn. Các chuẩn cấu trúc phải định ra các mục được thêm vào và phải định rõ các quy ước được sử dụng cho việc đánh số trang, tiêu đề, mục và mục con.

3- *Các chuẩn trình diễn tài liệu*: Các chuẩn trình diễn tài liệu định ra quy luật cho các tài liệu và đóng góp đáng kể cho tính kiên định của tài liệu. Chúng bao gồm việc định rõ các cỡ chữ và kiểu chữ được sử dụng trong tài liệu, cách sử dụng logo và các tên công ty, cách sử dụng màu sắc làm nổi bật cấu trúc của tài liệu,....

4- *Các chuẩn cập nhật tài liệu*: Khi soạn thảo tài liệu phản ánh các thay đổi trong hệ thống, chuẩn cập nhật tài liệu đóng vai trò chỉ thị phù hợp nhất cho hoạt động thay đổi tài liệu.

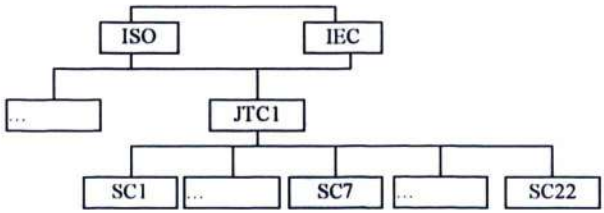
ISO 9001 dành cho việc tổ chức một giai đoạn thiết kế. Còn ISO 9000-3 là một hướng dẫn cài đặt 9001.

Một phân ban đặc biệt của ISO phụ trách về các chuẩn phát triển hoặc bảo trì phần mềm. Các chuẩn đó là ISO/IEC (hình 7.1) với phân nhánh JTC1/SC7, trong đó:

-JTC: Ban kỹ thuật,

-SC: Tiểu ban; SC7: Công nghệ thông tin hoặc là kỹ nghệ phần mềm,

-WG: Nhóm làm việc.



Hình 7.1. Sơ đồ các chuẩn quốc tế

Các chuẩn ISO đã công bố được bảo vệ bản quyền. Các tài liệu của ISO phải được mua trực tiếp từ ISO hoặc các tổ chức chuẩn hoá quốc tế.

ISO làm việc trên những tài liệu chuẩn. Tiến trình đo phần mềm dựa trên chuẩn WG13.

ISO 9126 được phát triển trên nền của ISO 9000.

7.2.1.2. Chuẩn ISO 9126

Chuẩn ISO 9126 ra đời nhằm các mục tiêu sau đây:

- Đặc tả tiêu chuẩn chất lượng sản phẩm

- Thiết lập các mục tiêu chất lượng sản phẩm

- Thiết lập các xét duyệt chất lượng kỹ thuật trước khi thực hiện cài đặt ở địa điểm của khách hàng.

Trong chuẩn ISO 9126, người ta đưa ra khái niệm khung chất lượng. Khung chất lượng đối với sản phẩm phần mềm gồm hai thành phần:

1-Các đặc tính chất lượng phù hợp theo những quan điểm về khung.

2-Sơ lược về độ đo đối với mỗi đặc tính trong từng quan điểm chất lượng phần mềm.

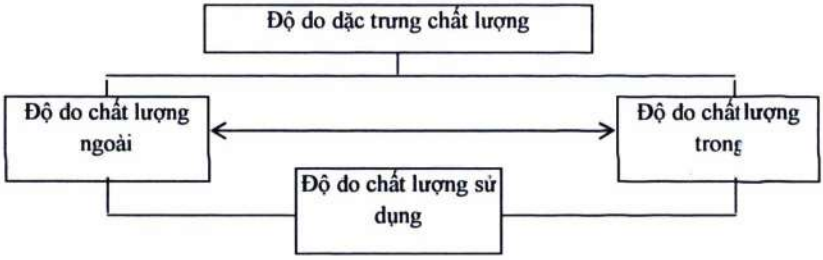
Trong chuẩn ISO 9126, người ta chi tiết hóa theo các phần:

9126-1 Phần 1- Mô hình chất lượng

9126-2 Phần 2- Độ đo chất lượng ngoài

9126-3 Phần 3- Độ đo chất lượng trong

9126-4 Phần 4- Độ đo chất lượng sử dụng.



Hình 7.2. Cấu trúc của các chuẩn WG6

Một số khái niệm sử dụng trong ISO 9126 gồm có:

• *Khái niệm "chất lượng" (quality):*

Chất lượng là toàn bộ các đặc tính của một sản phẩm liên quan tới khả năng thoả mãn các nhu cầu tường minh và ngầm định của sản phẩm.

Chất lượng sản phẩm phần mềm được đánh giá theo ba mặt:

1- Chất lượng ngoài (external quality)

2- Chất lượng trong (internal quality)

3- Chất lượng về mặt sử dụng (quality in use).

- *Chất lượng ngoài:* chất lượng dùng để đánh giá một sản phẩm thoả mãn các nhu cầu tường minh và ngầm định bên ngoài dưới những điều kiện đặc tả.

- *Chất lượng trong:* chất lượng toàn bộ các thuộc tính của một sản phẩm được sử dụng để xác định khả năng thoả mãn các nhu cầu tường minh và ngầm định bên trong của chúng dưới những điều kiện đặc tả.

- *Chất lượng về mặt sử dụng:* chất lượng dùng để đánh giá một sản phẩm có tính khả dụng thông qua người dùng chấp nhận những nhu cầu của họ nhằm đạt được các mục tiêu đã đặc tả mang tính hiệu lực, hiệu quả và tính thoả mãn trong một ngữ cảnh đặc tả về sử dụng.

• *Khái niệm "thuộc tính (attribute)":*

Thuộc tính là một tính chất trừu tượng hoặc vật lý và có thể đo được, đặc trưng riêng cho một thực thể.

• *Khái niệm "cách đo (measurement)":*

Cách đo là một quy trình gán một số hoặc một tính chất đặc thù vào một thực thể để mô tả một thuộc tính của thực thể đó.

• *Khái niệm "số đo (measure)":*

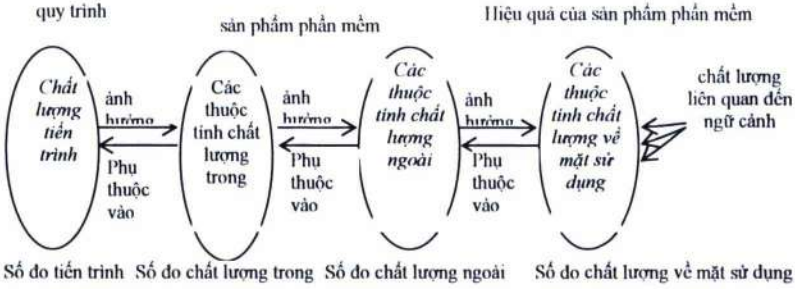
Số đo là số hoặc tính chất đặc thù gán vào một thuộc tính của một thực thể thông qua việc đo.

• *Khái niệm "số đo chất lượng trong"*

Số đo chất lượng trong là một số đo nhận được từ bản thân sản phẩm (mã chương trình, cấu trúc chương trình, cấu trúc CSDL, dữ liệu trong CSDL), hoặc trực tiếp hoặc gián tiếp.

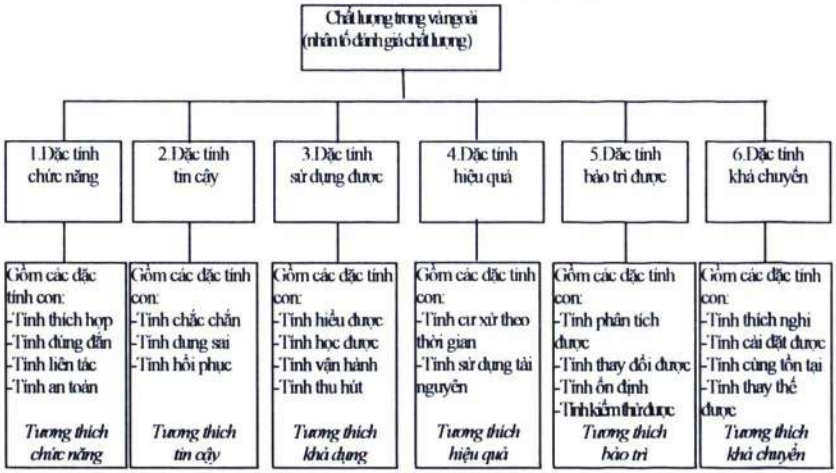
• *Khái niệm "số đo chất lượng ngoài"*

Số đo chất lượng ngoài là số đo gián tiếp về một sản phẩm được nhận từ số đo của hành vi của hệ thống cũng như hành vi của những thành phần hệ thống.



Hình 7.3.Chất lượng thể hiện trong vòng đời sản phẩm [18]

7.2.2. Mô hình chất lượng trong và chất lượng ngoài

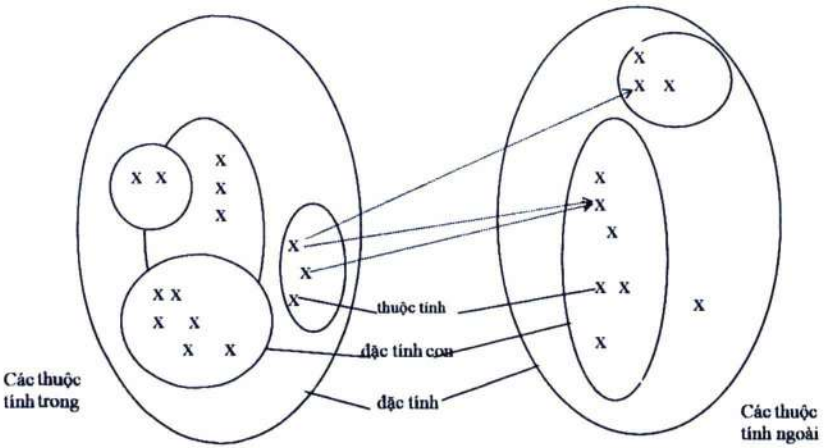


Hình 7.4. Sơ đồ phân rã chất lượng trong và ngoài [18]

7.2.3. Mô hình chất lượng về mặt sử dụng

Hình 7.5. Sơ đồ phân rã chất lượng về mặt sử dụng

7.2.4. Mô hình phân cấp đặc tính



Hình 7.6. Mô hình phân cấp các đặc tính chất lượng, các đặc tính con và các thuộc tính

7.2.5. Mô hình tiến trình phần mềm

Các mô hình tiến trình phần mềm tiêu biểu đã được trình bày trong chương 2 nhằm tạo ra một sản phẩm phần mềm có chất lượng cao.

Đối với từng loại hình công ty khác nhau, người ta sử dụng các chuẩn quản lý tiến trình chất lượng sản phẩm phần mềm *CMM (Capability Maturity Model)* và *CMMI (Capability Maturity Model Intergration)* của *SEI (Software Engineering Institute-Viện công nghệ phần mềm Mỹ)*. Đó là các phương pháp phát triển hay sản xuất ra các sản phẩm phần mềm, dùng để đánh giá và nâng cao năng lực phát triển phần mềm.

CMM và CMMI là một bộ khung (framework) những chuẩn đề ra cho một tiến trình sản xuất phần mềm hiệu quả. Các tổ chức áp dụng nó sẽ mang lại sự khả dụng về mặt chi phí, thời gian biểu, chức năng và chất lượng sản phẩm phần mềm.

Mô hình CMM mô tả các nguyên tắc và các thực tiễn nằm bên trong tính "thành thục" quá trình phần mềm và chủ ý giúp đỡ các công ty phần mềm hoàn thiện khả năng thuần thục quá trình sản xuất phần mềm, đi từ tự phát, hỗn độn tới các quá trình phần mềm có bài bản, có kỷ luật.

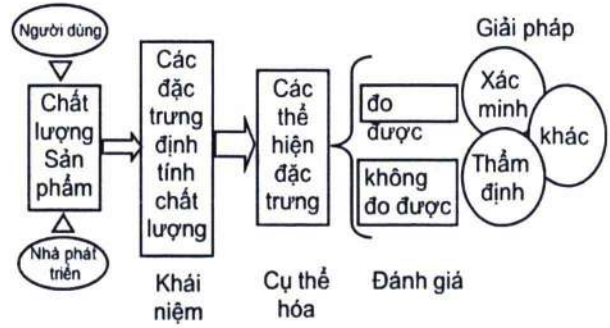
7.3. Việc đo chất lượng

7.3.1. Các chỉ tiêu đánh giá chất lượng phần mềm

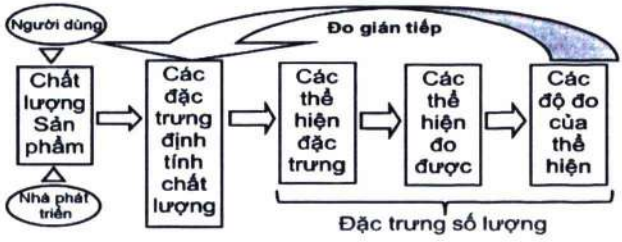
Các chỉ tiêu đánh giá chất lượng phần mềm có thể được phân loại thành hai nhóm chính:

1- Các chỉ tiêu có thể trực tiếp đo được (như lỗi/ nghìn dòng lệnh (KLOC)/ đơn vị thời gian).

2- Các chỉ tiêu chỉ có thể đo được một cách gián tiếp, như tính sử dụng hay tính bảo trì.



Hình 7.7. Đo các đặc tính chất lượng thông qua các thể hiện [12]



Hình 7.8. Đo gián tiếp các đặc trưng định tính [12]

Trong mỗi trường hợp phải có *cách đo*. Chúng ta phải so sánh phần mềm (tài liệu, chương trình và dữ liệu) với một cái *mốc* nào đó và đi tới một chỉ dẫn về chất lượng.

7.3.2. Các chỉ tiêu đánh giá cổ điển

Có một số chỉ tiêu đánh giá cổ điển điển hình [12] như:

a. Các chỉ tiêu đánh giá của McCall

McCall đã đề nghị một phân loại hữu ích về các chỉ tiêu đánh giá chất lượng phần mềm hội tụ vào ba khía cạnh (đặc tính) quan trọng của sản phẩm phần mềm:

1- Đặc trưng chức năng (nhân tố trực tiếp),

2- Khả năng đương đầu với những thay đổi (nhân tố gián tiếp),

3- Khả năng thích nghi với môi trường mới (nhân tố gián tiếp).

b.Các chỉ tiêu đánh giá chất lượng của Hewlett-Packard

Hewlett-Packard đã phát triển một tập các chỉ tiêu đánh giá chất lượng phần mềm được viết tắt là *FURPS (Functionality, Usability, Reliability, Performance, Sutrangortability)* tương ứng với đặc tính chức năng, tính khả dụng, tính tin cậy, tính thi hành và tính hỗ trợ.

Các chỉ tiêu đánh giá chất lượng và các thuộc tính FURPS có thể được dùng để thiết lập các độ đo chất lượng cho từng bước trong tiến trình kỹ nghệ phần mềm.

7.3.3. Các chỉ tiêu đánh giá chất lượng phần mềm theo ISO 9126

Chuẩn ISO /IEC 9126 được đưa ra vào năm 1991 quy định các chỉ tiêu đánh giá chất lượng phần mềm. Có 6 nhóm chỉ tiêu lớn (6 đặc tính đã được giới thiệu trong chương 1, phần *1.7.Tiêu chuẩn của một phần mềm tốt*) và được thể hiện thông qua 21 chỉ tiêu đánh giá con (đặc tính con). Sau đây là chi tiết hơn về các chỉ tiêu đã được dùng để đánh giá chất lượng phần mềm [18]:

Nhóm 1- Tính chức năng (Functionality)

Tính chức năng liên quan đến khả năng thực hiện một loạt các thao tác để đạt được các yêu cầu hiện hay ẩn của NSD. Nhóm này gồm các đặc tính con sau:

- Tính phù hợp (Suitability): *đặc tính của phần mềm chỉ ra các chức năng phải đầy đủ và phù hợp đối với các nhiệm vụ đã được xác định.*

- Tính chính xác (Accurateness): *đặc tính liên quan tới phần mềm phải cho kết quả hoặc hiệu quả chính xác như dự tính.*

- Tính liên tác (Interoperability): *đặc tính chỉ khả năng hoạt động tương tác giữa các hệ thống xác định.*

- Tính tuân thủ chuẩn (Compliance): *đặc tính làm cho phần mềm tham gia được với các ứng dụng liên quan tới các chuẩn, các quy ước, luật lệ.*

- Tính an toàn (Security): *khả năng dự phòng chống lại các truy cập không được phép dù là vô tính hay cố ý đối với chương trình hoặc dữ liệu.*

Nhóm 2- Tính tin cậy (Reliability)

Tính tin cậy là các nhân tố liên quan đến khả năng bảo toàn hiệu quả của phần mềm dưới những tác động của các điều kiện hay ở mức giới hạn nào đó. Nhóm này gồm các đặc tính con sau:

- Độ trưởng thành (Maturity): *thể hiện qua tỷ lệ trục trặc do lỗi của phần mềm.*

- Khả năng kháng lỗi (Faul Tolerance): *thể hiện khả năng bền vững của phần mềm, cho phép phần mềm vẫn duy trì được hiệu quả xác định dưới tác động của các yếu tố không thuận lợi như lỗi hay sự vi phạm về giao tiếp với phần mềm.*

- Khả năng hồi phục (Recoverability): *khả năng và nỗi lực cần thiết để khôi phục lại được hoạt động của hệ thống và khôi phục dữ liệu đạt được hiệu quả ở mức độ cho phép và kịp thời khi hệ thống gặp sự cố.*

Nhóm 3- Tính sử dụng được (Usability)

Tính sử dụng được liên quan khả năng sử dụng phần mềm theo hướng giảm thiểu những chi phí của NSD. Nhóm này gồm các đặc tính con sau:

- Dễ hiểu (Understandability): *đặc tính liên quan tới nỗ lực cần thiết của người dùng để hiểu được quan niệm logic của phần mềm cũng như khả năng áp dụng được những quan niệm đó.*

- Dễ học (Learnability): *đặc tính liên quan đến nỗ lực cần thiết của NSD để học cách sử dụng phần mềm.*

- Dễ thao tác (Operability): *đặc tính liên quan đến nỗ lực cần thiết của người sử dụng để thao tác và kiểm soát các thao tác đối với phần mềm.*

Nhóm 4- Tính hiệu quả (Efficiency)

Tính hiệu quả thể hiện khả năng của phần mềm về cung cấp hiệu năng yêu cầu, quan hệ với NSD tài nguyên trong điều kiện xác định. Nhóm này gồm các đặc tính con sau:

- *Hiệu quả thời gian (Time Behavior)*: thời gian đáp ứng, thời gian xử lý và tần xuất thông qua có hiệu quả (số lần thực hiện có tính chu kỳ trong một khoảng thời gian các chức năng của phần mềm).

- Hiệu quả tài nguyên (Resource Behavior)*: Khối lượng tài nguyên được sử dụng và khoảng thời gian sử dụng tài nguyên có hiệu quả.*

Nhóm 5- Tính bảo trì được (Maintenability)

Tính bảo trì được thể hiện khả năng của phần mềm có thể sửa chữa. Nhóm này gồm các đặc tính con sau:

- Phân tích được (Analysability): *đặc tính liên quan đến nỗ lực cần thiết để chẩn đoán khiếm khuyết hoặc nhận diện được phần cần phải thay đổi.*

- Thay đổi được (Changeability): *đặc tính liên quan đến nỗ lực cần thiết để sửa đổi, huỷ bỏ phần có lỗi hoặc thay đổi môi trường.*

- Ổn định (Stability)*: đặc tính liên quan đến các rủi ro hoặc các hiệu quả không mong muốn khi sửa đổi phần mềm.*

- Kiểm tra được (Testabilty)*: đặc tính liên quan đến các nỗ lực cần thiết để kiểm chứng tính hợp thức của phần mềm được thay đổi.*

Nhóm 6- Tính khả chuyển (Portability)

Tính khả chuyển thể hiện khả năng của phần mềm có thể chuyển từ một môi trường này sang môi trường khác. Nhóm này gồm các đặc tính con sau:

- Thích nghi được (Adaptability): *đặc tính thể hiện khả năng phần mềm thích ứng được với nhiều môi trường khác nhau mà không cần phải có những điều chỉnh đặc biệt.*

- Cài đặt được (Installability): *đặc tính liên quan đến các nỗ lực cần thết để cài đặt chương trình trong một môi trường xác định.*

- Tuân thủ (Conformance): *đặc tính thể hiện khả năng kết hợp được với các chuẩn và quy ước có liên quan khi thay đổi môi trường.*

- Có thể thay thế được (Replaceablity): *khả năng và các nỗ lực cần thiết để sử dụng phần mềm thay thế một phần mềm khác trong cùng môi trường của phần mềm.*

7.3.4. Các chỉ tiêu đánh giá chất lượng phần mềm khác

Các chỉ tiêu đánh giá chất lượng phần mềm khác là các chỉ tiêu cổ điển (đã được học trong giáo trình Kỹ nghệ phần mềm cơ sở ở cấp đại học) bao gồm [12]:

1- Độ đo chất lượng theo từng chỉ tiêu đánh giá;

2- Độ đo theo chỉ số chất lượng về cấu trúc thiết kế;

3- Độ đo khối lượng chương trình- mã nguồn của Halstead;

4- Độ đo mức độ phức tạp của McCabe;

5- Độ đo chất lượng bảo trì;

6- Độ đo bảo đảm chất lượng thống kê [24];

7- Đo độ tin cậy của phần mềm.

7.3.5. Quá trình phòng sạch

Quá trình *phòng sạch (clean room)*, do Mills đề xuất, là kiểm chứng chương trình một cách hình thức (chứng minh tính đúng đắn) kết hợp với việc bảo đảm chất lượng phần mềm thống kê.

Phương châm của kỹ thuật phòng sạch là phòng khiếm khuyết hơn trừ khiếm khuyết với cơ sở lý luận rằng: "*Nên dành nhiều thời gian cho kiểm chứng chương trình toán học hơn là cho gỡ lỗi*".

Với quá trình này, số khiếm khuyết trong mỗi KLOC giảm đáng kể: dự án phần mềm có kích cỡ từ 1000 đến 50000 LOC, dùng công nghệ phòng sạch tìm thấy 90% khiếm khuyết trước khi kiểm thử lần đầu.

Hạn chế lớn nhất khi áp dụng kỹ thuật này là phạm vi hoạt động hẹp vì đa số hệ thống phần mềm, chỉ một phần nhỏ có thể đặc tả hình thức.

7.4. Các hoạt động bảo đảm chất lượng

7.4.1. Thẩm định và xác minh

Thẩm định (Validation) và *xác minh (Verification)* một hệ phần mềm là các hoạt động nổi trội nhất bảo đảm chất lượng phần mềm. Đó là quá trình liên tục xuyên suốt mọi giai đoạn của quá trình phần mềm. Thẩm định và xác minh mang tính quá trình nhằm bảo đảm phần mềm thoả mãn các yêu cầu của khách hàng. Như thế, thẩm định và xác minh là một tiến trình kéo dài suốt vòng đời. Nó bắt đầu từ khi duyệt xét yêu cầu phần mềm.

Các loại yêu cầu phần mềm gồm có:

1-Yêu cầu người dùng:

+Yêu cầu chức năng *(số lượng, mô tả)*

+Yêu cầu phi chức năng *(đo được)*

+Yêu cầu miền ứng dụng *(chức năng và quy tắc nghiệp vụ).*

2-Yêu cầu người phát triển:

+Yêu cầu hệ thống: *các đặc trưng hệ thống*

+Yêu cầu *môi trường, công nghệ và công cụ phát triển.*

Đặc tả yêu cầu được tiến hành với nhiều lý do, trong đó có hai lý do chính nổi bật. Thứ nhất, đó là cơ sở để phát triển phần mềm. Thứ hai, đó là cơ sở cho việc xác minh lại phần mềm. Hơn thế, chúng ta cần xác minh để bảo đảm phần mềm bám sát đặc tả.

Thẩm định và xác minh nhằm vào hai mục tiêu:

1- Phát hiện các khuyết tật trong hệ thống.

2- Đánh giá xem hệ thống có dùng được hay không.

Sự khác nhau quan trọng giữa thẩm định và xác minh theo nhận xét của Boehm [25]:

Thẩm định: kiểm tra xem sản phẩm phần mềm được xây dựng có phải là sản phẩm đúng không ?

Xác minh: kiểm tra xem sản phẩm phần mềm được xây dựng có đúng là sản phẩm được yêu cầu không ?

Như vậy, thẩm định là kiểm tra xem chương trình có được như mong đợi của người dùng hay không, tức là chương trình có hoạt động đúng không. Còn xác minh là kiểm tra xem chương trình có phù hợp với đặc tả yêu cầu hay không.

Cơ sở để thẩm định phần mềm *(cái để đối chiếu)* bao gồm:

- Các đặc tả yêu cầu;
- Các thiết kế *(nếu có);*
- Các chuẩn *(sử dụng).*

Các hoạt động thẩm định gồm:

- Rà soát;
- Kiểm toán;
- Kiểm thử của người dùng.

Cơ sở xác minh bao gồm:

- Các yêu cầu chức năng;
- Các yêu cầu phi chức năng *(đo được);*
- Các yêu cầu khác *(chuẩn, công nghệ, công cụ, sự mong đợi của người dùng).*

Các hoạt động xác minh gồm:

- Rà soát;
- Đối chiếu với đặc tả *(đầy đủ các thành phần, chính xác, không mâu thuẫn).*

7.4.2. Hoạt động thực hiện kiểm thử

Kiểm thử phần mềm (software testing) là một trong những yếu tố góp phần *bảo đảm chất lượng phần mềm (SQA)*, là khâu điển hình kiểm soát đặc tả, thiết kế, lập mã.

Theo Glen Myers [23]: *"Kiểm thử phần mềm là quá trình vận hành chương trình để tìm ra lỗi".*

Kiểm thử phần mềm được đặt ra với những lý do:

- Muốn nhận diện phần mềm như một phần tử của hệ thống hoạt động *(xem xét sản phẩm)*

- Hạn chế chi phí cho các thất bại do lỗi gây ra sau này *(hiệu quả)*

- Có kế hoạch tốt nâng cao chất lượng suốt quá trình phát triển *(giải pháp)*.

Kiểm thử giữ vai trò lớn trong quá trình phát triển phần mềm. Xét theo tiêu chí về chi phí thì kiểm thử chiếm:

- 40% tổng công sức phát triển;

- ≥ 30% tổng thời gian phát triển;

- Với các phần mềm có ảnh hưởng tới sinh mạng, chi phí có thể gấp từ 3 đến 5 lần tổng các chi phí khác cộng lại.

Như vậy, kiểm thử tốt sẽ:

- Giảm chi phí phát triển;

- Tăng độ tin cậy của sản phẩm phần mềm.

Vấn đề đặt ra là cần vận hành phần mềm như thế nào để:

- Hiệu suất tìm ra lỗi là cao nhất ?

- Chi phí (*thời gian, công sức*) ít nhất?

Công việc trước mắt của kiểm thử phần mềm là tạo ra các ca kiểm thử (mạch thực hiện trong chương trình) để tìm ra lỗi của phần mềm.

Mục đích cuối cùng của kiểm thử phần mềm nhằm có một chương trình tốt, chi phí ít.

Glen Myers phát biểu một số quy tắc giống như mục đích kiểm thử:

Quy tắc 1- Kiểm thử là một tiến trình thực hiện một chương trình với ý định tìm ra lỗi.

Quy tắc 2- Một ca kiểm thử là một trường hợp kiểm thử có xác suất cao *để tìm ra lỗi.*

Quy tắc 3-Việc kiểm thử thành công *là việc kiểm thử làm lộ ra một lỗi còn chưa được phát hiện.*

Các mục đích trên dẫn đến một sự thay đổi lớn trong quan điểm. Chúng đi ngược lại quan điểm thông thường là một phép kiểm thử thành công là kiểm thử không tìm ra lỗi nào. Mục đích của chúng ta là thiết kế các phép kiểm thử để làm lộ ra một cách có hệ thống những lớp lỗi khác nhau và làm như vậy với một số lượng thời gian và công sức ít nhất.

Nếu kiểm thử được tiến hành thành công (theo mục tiêu được phát biểu trên), *thì nó sẽ làm lộ ra những lỗi trong phần mềm*. Việc kiểm thử phần mềm làm việc theo đặc tả nên các yêu cầu hiệu năng dường như là được đáp ứng. Bên canh đó, dữ liệu thu thập được khi việc kiểm thử được tiến hành đưa ra một chỉ dẫn tốt về độ tin cậy phần mềm và một chỉ dẫn nào đó về phẩm chất phần mềm với tư cách toàn cục.

Có một điều mà kiểm thử không thể làm được: *Kiểm thử không thể chứng minh được việc không có khiếm khuyết, nó chỉ có thể chứng minh được khiếm khuyết phần mềm hiện hữu. (*Điều quan trọng là cần nhớ điều này trong tâm người thực hiện kiểm thử).

Khi kiểm thử, người ta đưa ra những khái niệm về ca kiểm thử "tốt" và "thắng lợi":

- Ca kiểm thử *tốt* là ca kiểm thử có xác suất cao tìm ra 1 lỗi.

- Ca kiểm thử *thắng lợi* là ca kiểm thử làm lộ ra ít nhất 1 lỗi.

Vấn đề đặt ra ở chỗ nếu không tìm được lỗi nào thì có thể kết luận phần mềm hoàn hảo? Câu trả lời chung là chưa hẳn như vậy.

Kiểm thử có nhiều lợi ích, trong đó phải kể đến các lợi ích quan trọng:

- Ca kiểm thử thắng lợi làm lộ ra khiếm khuyết

- Kiểm thử mang lại các lợi ích phụ là thuyết minh:

+ Chức năng tương ứng với đặc tả,

+ Thực thi phù hợp yêu cầu và đặc tả,

+ Cung cấp các chỉ số độ tin cậy và chất lượng.

Tuy kiểm thử có nhiều lợi ích như trên nhưng chưa thể khẳng định phần mềm không còn khiếm khuyết.

Chúng ta sẽ nghiên cứu sâu hơn về kiểm thử phần mềm trong chương 8 về mô hình chiến lược kiểm thử, mức kiểm thử, phương pháp và kỹ thuật kiểm thử, thiết kế ca kiểm thử, tiến trình kiểm thử và các công cụ kiểm thử.

7.4.3. Quản lý cấu hình phần mềm

Mục đích của *quản lý cấu hình phần mềm* (*Software Configuration Management*-SCM) là *thiết lập và bảo đảm tính toàn vẹn của các sản phẩm* trung gian cũng như sản phẩm cuối cùng của một dự án phần mềm trong suốt vòng đời của dự án.

Pressman xem SCM như một hoạt động bảo đảm chất lượng phần mềm. Nó bao trùm suốt tiến trình phát triển và tiến hóa của phần mềm.

Các khoản mục cấu thành lên các thành phần phần mềm được tạo ra như là những chế tác *(artifact)* của tiến trình kỹ nghệ phần mềm được tập hợp lại với một tên chung gọi là *cấu hình phần mềm* (*software configuration - SC*).

Các chế tác này có nhiều mức khác nhau:

- Bộ phận - tổng thể (*phạm vi*).

- Chưa hoàn thiện - hoàn thiện (*theo tiến trình, chất lượng*).

- Ở các mức tiến hóa khác nhau (*các phiên bản - version*).

Kết quả của tiến trình kỹ nghệ phần mềm là 3 chế tác:

- Các chương trình máy tính (mức nguồn và mức mã máy).

- Các tài liệu mô tả chương trình máy tính (nhắm đến cả những nhà phát triển lẫn những người dùng).

- Các cấu trúc dữ liệu (cả bên trong và ngoài chương trình).

Cấu hình phần mềm là những bộ phận hay toàn bộ của các thành phần này nhận được khi phát triển.

Quản lý cấu hình phần mềm là tập các hoạt động được phát triển để quản lý sự thay đổi của phần mềm trong suốt vòng đời phần mềm.

Quản lý cấu hình phần mềm bao quát các nội dung (đã được học trong giáo trình Kỹ nghệ phần mềm cơ sở, cấp đại học) dưới đây [12]:

1- Quản lý mốc giới (*Baseline Management*)

2- Xác định các khoản mục cấu hình (*Configuration Items*)

3- Kiểm soát phiên bản (*Version Contrrol*)

4- Kiểm soát thay đổi (*Change Contrrol*)

5- Kiểm toán cấu hình (*Configuration Review*)

6- Bảo cáo hiện trạng cấu hình (*Configuration Status Accounting*).

7.4.4. Các hoạt động khác bảo đảm chất lượng phần mềm

Trong quá trình phát triển phần mềm, các nhân tố khác với các nhân tố đã kể trên cũng ảnh hưởng thực sự lên chất lượng mà không phải là yêu cầu. Mặt khác, công nghệ, công cụ và quản lý thực sự cũng ảnh hưởng lên chất lượng. Một số hoạt động SQA tiêu biểu (đã được học trong giáo trình Kỹ nghệ phần mềm cơ sở, cấp đại học [12]), như:

1- Áp dụng công nghệ, kỹ nghệ hiệu quả;

2- Rà soát kỹ thuật chính thức và rà soát phần mềm;

3- Tuân theo các chuẩn phát triển;

4- Khống chế thay đổi;

5- Thực hiện đo lường;

6- Báo cáo và quản lý các báo cáo;

7- Gỡ lỗi;

8- Phân tích an toàn phần mềm.

Để có hiệu quả, cần phải phân tích phần mềm trong ngữ cảnh của toàn hệ thống. Có thể dùng các kỹ thuật phân tích như "phân tích cây lỗi", logic thời gian thực, lưới PETRY.

7.5. Tiến trình triển khai bảo đảm chất lượng

Tiến trình triển khai bảo đảm chất lượng phần mềm gồm các giai đoạn:

1- Khảo sát nhu cầu;

2- Lập kế hoạch;

3- Triển khai.

7.5.1. Khảo sát nhu cầu

Mọi tổ chức phát triển phần mềm đều có cơ chế riêng để đánh giá chất lượng.

Ở mức thấp nhất, chất lượng là trách nhiệm của từng cá nhân các kỹ sư, người rà soát, người kiểm thử ở từng mức thích hợp.

Ở mức cao nhất có nhóm SQA chuyên môn hoá:

- Thiết lập các chuẩn và thủ tục để đạt chất lượng;
- Thực thi trách nhiệm bảo đảm chúng được tuân thủ.

Câu hỏi cho mọi tổ chức khi bắt đầu: Ta đang ở đâu?

Trước hết, mọi tổ chức phát triển phần mềm cần chấp nhận các thủ tục, phương pháp, công cụ của kỹ nghệ phần mềm (các nhân tố kỹ thuật).

Tiếp theo, cố gắng để tạo ra các thủ tục bảo đảm chất lượng phần mềm là một kiểm toán SQA/SCM.

Đánh giá hiện trạng SQA và *SCM (quản lý cấu hình phần mềm)* của tổ chức thông qua khảo sát ba chủ đề sau:

1- Kiểm kê các chính sách SQA: chính sách, thủ tục, chuẩn nào đã có trong các pha phát triển?;

2- Đánh giá vai trò của kỹ nghệ phần mềm, SQA trong tổ chức hiện tại có hiệu lực đến đâu?;

3- Đánh giá mối quan hệ giữa SQA với các đơn vị khác: các người thực hiện rà soát kỹ thuật chính thức, quản lý cấu hình và kiểm thử.

Khi ba câu hỏi trên đã được trả lời thì mức độ mạnh hay yếu đã được xác định.

Nếu có nhu cầu SQA thì cần tiến hành đánh giá thận trọng bằng quy tắc bỏ phiếu.

7.5.2. Lập kế hoạch triển khai

Việc lập kế hoạch triển khai SQA cần dựa trên các chuẩn, trong đó đặc biệt là các chuẩn sau:

- *DOD-STD-2167A: công nghệ phần mềm*
- *DOD-STD-2168: đánh giá chất lượng phần mềm*
- *FAA-STD-018: chuẩn SQA cho FAA*
- *IEEE Sst. 730-1984: kế hoạch SQA*
- *IEEE Sst. 983-1986: lập kế hoạch SQA*
- *IEEE Sst. 1028-1988: kiểm toán và rà soát phần mềm*
- *IEEE Sst. 1012-1986: kế hoạch kiểm thử và thẩm định phần mềm.*

Theo ANSI/IEEE STANDARDS 30-1984 và 983-1986, kế hoạch SQA bao gồm:

- *Mục đích của kế hoạch*
- *Tài liệu tham khảo*
- *Quản lý:*

+ Tổ chức

+ Các nhiệm vụ

+ Các trách nhiệm

- Nội dung kế hoạch SQA

- Tài liệu:

+ Mục đích

+ Tài liệu công nghệ phần mềm được yêu cầu

+ Các tài liệu khác

- Các chuẩn, các luật và các quy ước:

+ Mục đích

+ Các quy ước

- Kế hoạch rà soát và kiểm toán

+ Mục đích

+ Các nội dung rà soát:

- Rà soát yêu cầu phần mềm
- Rà soát thiết kế
- Rà soát việc xác minh và thẩm mịnh
- Kiểm toán chức năng

• Kiểm toán vật lý

• Kiểm toán bên trong quá trình

• Kiểm toán quản lý.

- Kế hoạch khác SQA:

+ Quản lý cấu hình phần mềm

+ Báo cáo và chỉnh sửa

+ Kiểm soát mã

+ Kiểm soát phương tiện

+ Kiểm soát nhà cung cấp

+ Thu thập, bảo trì và bảo quản các báo cáo

7.5.3. Triển khai

Triển khai SQA có những vấn đề sau:

- Khó thiết lập trong một tổ chức nhỏ (vì thiếu nguồn lực để thực hiện các hoạt động cần thiết);

- Cần sự thay đổi mang tính văn hoá: không dễ dàng thực hiện;

- Tốn không ít công sức và tiền của.

7.5.3.1. Nguyên tắc triển khai SQA

Nguyên tắc Chi phí: ở mức cơ bản SQA được xem là hiệu quả về chi phí nếu:

$$C3 > C1 + C2$$

Ở đây:

- C3 là chi phí từ các sai do không có SQA;

- C1 là chi phí cho tổ chức, vận hành SQA;

- C2 là chi phí do các sai không tìm thấy khi đã có SQA.

7.5.3.2 Quy trình đánh giá chất lượng phần mềm theo chuẩn 14598 [18]

Bước1. Lập kế hoạch đánh giá chất lượng phần mềm

Mục tiêu: Một kế hoạch chuẩn hoá cho việc quản lý các đánh giá sản phẩm phần mềm.

Nội dung:

+ Định dạng các nhu cầu cần đánh giá về các mặt:

- Tổ chức;

- Quản lý dự án;

- Kế hoạch hoá các nhu cầu: kế hoạch đo.

+ Cân nhắc chung, các tiến trình lập kế hoạch đo, các cân nhắc chi tiết.

Quản lý ở mức tổ chức:

- Lập kế hoạch việc sử dụng và cải thiện công nghệ đánh giá;

- Cài đặt công nghệ đánh giá;

- Chuyển giao kỹ thuật đã thường sử dụng để đánh giá;

- Đánh giá về mặt công nghệ đã sử dụng để đánh giá;

- Quản lý các kinh nghiệm.

Hỗ trợ quản lý dự án:

- Hỗ trợ cho kế hoạch hoá đánh giá;

- Tiếp tục đẩy mạnh kế hoạch đánh giá chất lượng;

- Ủng hộ các dự án đánh giá;

- Chọn lọc các kết quả đánh giá.

Dàn bài lập kế hoạch đánh giá thường có những phần sau:

Chương1: Giới thiệu (mục đích, tiếp kiến của kế hoạch và dự kiến ứng dụng của kế hoạch

Chương 2: Các mục tiêu đánh giá

Chương 3: Các đặc tính chất lượng vận dụng

Chương 4: Danh sách ưu tiên

Chương 5: Các mục tiêu chất lượng (các đặc tính)

Chương 6: Kế hoạch thời gian

Chương 7: Trách nhiệm và bổn phận của những người tham gia dự án

Chương 8: Các phân loại đo (đo chất lượng sản phẩm , đo chất lượng quy trình)

Chương 9: Việc sử dụng và phân tích dữ liệu

Chương 10: Lập báo cáo

Chương 11: Các yêu cầu khác.

Bước2. Thực hiện và quản lý việc đánh giá chất lượng phần mềm

- Thực hiện Quy trình của người triển khai

Các yêu cầu của quy trình đánh giá gồm:

1- Các yêu cầu chung (các yêu cầu thuộc về tổ chức và dự án);

2- Xác lập các yêu cầu đánh giá (các yêu cầu chất lượng);

3- Đặc tả về đánh giá (Các yêu cầu chất lượng trong và ngoài;

4- Thiết kế các đánh giá (lập kế hoạch đánh giá trong và ngoài);

5- Tiến hành đánh giá (đánh giá trong, đánh giá sản phẩm cuối);

6- Phản hồi tới tổ chức.

Quy trình đánh giá gồm các pha:

1-Thiết kế đánh giá:

- Thiết lập việc thu thập dữ liệu cho các độ đo;
- Lập các quyết định đặc tả đã yêu cầu;
- Đặc tả các tiêu chuẩn đã có.

2-Thực hiện đánh giá:

- Thực hiện cho từng thiết kế;
- Thực hiện các phương pháp đo hợp lệ;
- Viết tài liệu;
- Thông qua các chỉ tiêu chính;
- So sánh với mục tiêu;
- So sánh với các dự án khác.

3-Phân tích kết quả đánh giá (khi phát triển đã hoàn thành):

- Phân tích các kết quả đánh giá;

- Cải tiến quy trình đánh giá và các độ đo, việc đo;

- Khẳng định phương thức hợp lệ đã đúng.

- *Thực hiện Quy trình đối với người mua*

Các dữ liệu vào phục vụ quy trình đánh giá gồm có:

- Những yêu cầu hệ thống;

- Những yêu cầu ở mức toàn vẹn;

- Đặc tả yêu cầu phần mềm;

- Dữ liệu đánh giá được thực hiện bởi các tiến trình khác.

Chất lượng của một đánh giá theo quan điểm của người mua tập trung vào các điểm chính sau đây:

- Khả năng lặp lại;

- Khả năng tái tạo;

- Tính công bằng;

-Mục tiêu.

Đánh giá tổng thể dựa trên quy trình đánh giá các tỷ trọng ước lượng về:

Bước 1- Đánh giá về Chức năng;

Bước 2- Đánh giá về Chất lượng;

Bước 3- Đánh giá về Chi phí;

Bước 4- Đánh giá về Mạo hiểm;

Bước 5- Đánh giá về những vấn đề Khác.

Đánh giá tổng thể chính là đánh giá tổng hợp dựa theo 5 bước trên.

-Thực hiện Quy trình đối với người đánh giá chất lượng phần mềm

Mục tiêu của quy trình đối với người đánh giá chất lượng phần mềm là chuẩn hoá một quy trình đánh giá sản phẩm phần mềm.

Trong quy trình, cần xác định:

1- Các phần chưa đánh giá;

2- Các đặc tính của tiến trình đánh giá;

3- Tiến trình đánh giá sản phẩm phần mềm và các tính năng;

4- Cái vào và cái ra;

5- Mối quan hệ trong vòng đời phát triển phần mềm.

Các trách nhiệm của người yêu cầu đánh giá bao gồm:

- Lập bản quyền trong sản phẩm phần mềm;

- Cung cấp thông tin về sản phẩm;

- Cung cấp những yêu cầu đánh giá ban đầu về trạng thái, những yêu cầu đánh giá;

- Cung cấp những yêu cầu tin cậy về trạng thái;

- Hoạt động lập tức khi có yêu cầu;

- Cung cấp cách truy nhập thích hợp vào máy tính và tài nguyên;

- Bảo đảm cung cấp kịp thời về thông tin;

- Cung cấp thông tin đánh giá về một vài thông tin có thể hợp lệ.

Các trách nhiệm của người làm nhiệm vụ đánh giá bao gồm:

- Kiểm tra lại các bản quyền;

- Giữ cẩn mật, bao gồm cả ghi âm;

- Cung cấp các nghề nhân viên chất lượng đã được đào tạo cho đánh giá;

- Cung cấp các công cụ đánh giá và công nghệ;

- Hướng dẫn đánh giá thích hợp với yêu cầu;

- Bảo đảm phân phát đánh giá ngay lập tức;

- Cung cấp những hướng dẫn rõ ràng khi đánh giá.

Các nội dung yêu cầu đánh giá:

- Mục đích của đánh giá;

- Quy mô theo dõi (các đặc tính chất lượng, các đặc tính con, quan hệ quan trọng, thành phần, tài liệu, độ đo,...);

- Quy mô tin cậy và độ chính xác đánh giá;

- Kết nối thuận tiện các yêu cầu đánh giá.

TÓM TẮT CHƯƠNG

Chất lượng là khả năng đáp ứng toàn diện nhu cầu của người dùng về tính năng cũng như cũng như công dụng được nêu ra một cách tường minh hoặc không tường minh trong những ngữ cảnh xác định.

Bảo đảm chất lượng phần mềm (Software Quality Assurance-SQA) là một hoạt động được áp dụng trong toàn bộ tiến trình kỹ nghệ phần mềm, bao phủ các quá trình phân tích, thiết kế, mã hóa, kiểm thử, các cuộc họp chính thức, các chiến lược kiểm thử, quá trình kiểm soát tư liệu, sự tuân thủ các chuẩn, cơ chế đo đạc và báo cáo.

Các chuẩn của ISO về phần mềm giúp chúng ta quản lý chất lượng theo cách chuyên nghiệp nhất.

Trong chuẩn ISO 9126, người ta đưa ra khái niệm khung chất lượng. Khung chất lượng đối với sản phẩm phần mềm gồm hai thành phần:

1-Các đặc tính chất lượng phù hợp theo những quan điểm về khung;

2-Sơ lược về độ đo đối với mỗi đặc tính trong từng quan điểm chất lượng phần mềm.

Trong chuẩn ISO 9126, người ta chi tiết hóa theo các Mô hình chất lượng, Độ đo chất lượng ngoài, Độ đo chất lượng trong và Độ đo chất lượng sử dụng.

Một quy trình phần mềm là một dãy được sắp các hoạt động để tạo ra một sản phẩm phần mềm. Mặc dù có nhiều quy trình phần mềm, nhưng tồn tại một số hoạt động cơ bản giống nhau với tất cả các quy trình.

Các chỉ tiêu đánh giá chất lượng phần mềm có thể được phân loại thành hai nhóm chính là 1-trực tiếp đo được và 2-có thể gián tiếp đo được.

Chuẩn ISO /IEC 9126 được đưa ra vào năm 1991 quy định các chỉ tiêu đánh giá chất lượng phần mềm. Có 6 nhóm đặc tính lớn và được thể hiện thông qua 21 đặc tính con.

Có nhiều hoạt động như thẩm định và xác minh, kiểm thử, quản lý cấu hình phần mềm và nhiều hoạt động khác được tiến hành để bảo đảm chất lượng phần mềm.

Tiến trình triển khai bảo đảm chất lượng phần mềm thường chia làm 3 giai đoạn:

1-Khảo sát nhu cầu;

2-Lập kế hoạch;

3-Triển khai.

Quy trình đánh giá chất lượng phần mềm theo chuẩn 14598 gồm 2 bước chính: 1-Lập kế hoạch đánh giá chất lượng phần mềm, 2-Thực hiện và quản lý việc đánh giá chất lượng phần mềm.

CÂU HỎI CHƯƠNG 7

7.1. Anh/Chị hãy đưa ra một ví dụ về các đặc tính chất lượng, các đặc tính con và các thuộc tính?

7.2. Tất cả các chỉ tiêu đánh giá chất lượng phần mềm có thể trực tiếp đo được không?

7.3. Các chỉ tiêu đánh giá chất lượng phần mềm theo ISO 9126 có gì mới so với các chỉ tiêu đánh giá cổ điển?

7.4. Tư tưởng bảo đảm chất lượng thống kê là gì?

7.5. Anh/Chị hiểu thế nào là quá trình phòng sạch?

7.6. Thẩm định và xác minh có gì giống nhau, có gì khác nhau?

7.7. Thực chất của kiểm thử phần mềm là gì? Kiểm thử có thể chứng minh được việc phần mềm không có khiếm khuyết?

7.8. Tại sao phải quản lý cấu hình phần mềm?

7.9. Nguyên tắc chi phí triển khai SQA?

7.10. Quy trình đánh giá chất lượng phần mềm theo chuẩn 14598 gồm mấy bước; đó là những bước nào?

BÀI TẬP CHƯƠNG 7

Bài tập 1.1

Trong bảng mô tả đối tượng cấu hình phần mềm dưới đây, các khoản mục cấu hình nào có dạng tài liệu?

TT	Khoản mục cấu hình	Mã cấu hình	Dạng (Tài liệu/dữ liệu/chương trình)
1	Kế hoạch dự án	KHDA_TVH0722_x.x_ddmmyy	
2	Tiến trình chi tiết	TTCT_TVH0722_x.x_ddmmyy	
3	Đặc tả nghiệp vụ	ĐTNV_TVH0722_x.x_ddmmyy	
4	Đặc tả yêu cầu	ĐTYC_TVH0722_x.x_ddmmyy	
5	Đặc tả Thiết kế	ĐTTK_TVH0722_x.x_ddmmyy	
6	Đặc tả Kiểm thử	ĐTKT_TVH0722_x.x_ddmmyy	
7	Dữ liệu input	DLINPUT_TVH0722_x.x_ddmmyy	
8	Mã nguồn	MANGUON_TVH0722_x.x_ddmmyy	

Bài tập 1.2

Xác định quan hệ giữa các đối tượng cấu hình?

(biểu diễn mối quan hệ Tổng thể- bộ phận bằng mũi tên đứt nét một chiều, mối quan hệ Ngang hàng bằng mũi tên liền nét hai chiều)

Chương 8
KIỂM THỬ PHẦN MỀM

Kiểm thử phần mềm (software testing) là một trong những yếu tố góp phần bảo đảm chất lượng phần mềm, là khâu điển hình kiểm soát đặc tả, thiết kế, lập mã.

Chương này trình bày chi tiết các nội dung về kiểm thử phần mềm.

Học viên cần nắm vững các khái niệm và mô hình chiến lược kiểm thử, phân biệt được các mức kiểm thử, biết vận dụng các phương pháp và kỹ thuật kiểm thử, rèn luyện kỹ năng thiết kế ca kiểm thử.

Học viên cũng cần tuân theo tiến trình kiểm thử và cần biết một số công cụ kiểm thử phổ biến.

8.1. Chiến lược kiểm thử

8.1.1. Khái niệm chiến lược kiểm thử

Chiến lược kiểm thử là sự tích hợp các kỹ thuật thiết kế "ca kiểm thử" tạo thành một dãy các bước nhằm hướng dẫn quá trình kiểm thử phần mềm thành công. *Ca kiểm thử (test case)* là một tình huống kiểm thử tương ứng với một mạch hoạt động của chương trình. Nó bao gồm một tập các giá trị đầu vào và một danh sách các kết quả đầu ra mong muốn và thực tế.

Chiến lược kiểm thử được đặt ra với mục tiêu nhằm phác thảo lộ trình để:

- Nhà phát triển tổ chức việc bảo đảm chất lượng bằng kiểm thử;
- Khách hàng hiểu được công sức, thời gian và nguồn lực cần cho kiểm thử.

Chiến lược kiểm thử cần phải đạt những yêu cầu sau:

- *Tích hợp được các khâu như lập kế hoạch, thiết kế ca kiểm thử, tiến hành kiểm thử, thu thập và đánh giá các thông tin kết quả;*
- *Đủ mềm dẻo để cổ vũ óc sáng tạo, đáp ứng được yêu cầu khách hàng;*

- *Thích ứng với mức kiểm thử cụ thể;*
- *Đáp ứng các đối tượng quan tâm khác nhau.*

8.1.2. Mô hình chiến lược tổng thể

Về mặt kỹ nghệ, việc kiểm thử gồm một số bước được thực hiện tuần tự.

Ban đầu, việc kiểm thử tập trung vào từng mô-đun riêng biệt bảo đảm rằng nó vận hành đúng đắn như một đơn vị. Do đó mới có tên kiểm thử đơn vị. Kiểm thử đơn vị dùng rất nhiều các kỹ thuật kiểm thử hộp trắng, kiểm soát các đường đặc biệt trong cấu trúc điều khiển của một mô-đun nhằm phát hiện tối đa các lỗi. Mặt khác, các mô-đun phải được lắp ghép hay tích hợp lại để tạo nên phần mềm hoàn chỉnh.

Việc kiểm thử tích hợp có liên quan tới thẩm định và xây dựng chương trình. Các kỹ thuật thiết kế kiểm thử hộp đen được dùng trong hầu hết quá trình tích hợp, mặc dầu các kiểm thử hộp trắng cũng có thể được dùng để bao quát đa số các đường điều khiển.

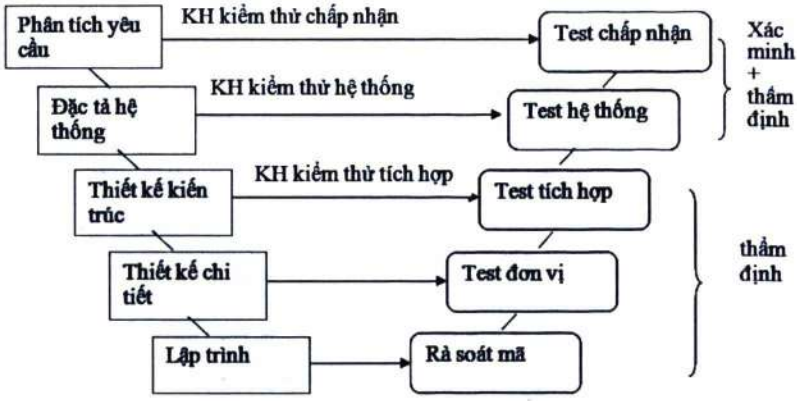
Sau khi phần mềm đã được tích hợp (được xây dựng), một tập các phép kiểm thử sẽ được tiến hành. Các tiêu chuẩn hợp lệ (được thiết lập trong phân tích yêu cầu) cũng phải được kiểm thử.

Việc kiểm thử hợp lệ được tiến hành nhằm bảo đảm phần mềm đáp ứng đầy đủ tất cả các yêu cầu chức năng. Các kỹ thuật kiểm thử hộp đen được dùng chủ yếu trong kiểm thử hợp lệ.

Kiểm thử hệ thống nằm trong khung cảnh rộng hơn của kỹ nghệ hệ thống máy tính. Khi làm hợp lệ, phần mềm phải được tổ hợp với các phần tử hệ thống khác (như phần cứng, con người, CSDL). Vì vậy kiểm thử hệ thống là rất quan trọng.

Cuối cùng, kiểm thử chấp nhận *sẽ thẩm định lại rằng tất cả các thành phần có khớp với nhau không, cũng như chức năng hay độ hoàn thiện của hệ thống có đạt được không.*

Mô hình chiến lược kiểm thử tổng thể được đề xuất theo hình chữ V như sau:



Hình 8.1. Mô hình chiến lược kiểm thử tổng thể [12]

8.1.3. Một số chiến lược kiểm thử bổ trợ

Ngoài chiến lược kiểm thử tổng thể, người ta còn tiến hành một loạt các chiến lược kiểm thử bổ trợ như [12]:

- Kiểm thử hệ thời gian thực;
- Kiểm thử Alpha và Beta;
- Kiểm thử so sánh.

8.1.3.1. Chiến lược kiểm thử hệ thời gian thực

Hệ thời gian thực là hệ thống đáp ứng đúng, chính xác các sự kiện của môi trường.

Kiểm thử hệ thống thời gian thực là rất khó. Những người thiết kế ca kiểm thử *không chỉ phải xem xét các trường hợp kiểm thử hộp đen và hộp trắng mà còn phải xem xét cả việc chia thời gian cho dữ liệu và cơ chế song song cho các nhiệm vụ (tiến trình) giải quyết dữ liệu đó.* Trong nhiều tình huống, trạng thái của hệ thống cũng có thể dẫn tới lỗi.

Mối quan hệ mật thiết giữa phần mềm thời gian thực và môi trường phần cứng của nó cũng có thể gây ra các vấn đề cho kiểm thử. Việc kiểm thử phần mềm phải xem xét tác động của các lỗi phần cứng đến xử lý phần mềm. Những lỗi như thế rất khó mô phỏng sát thực tế.

Để khắc phục khó khăn trên, chiến lược kiểm thử được vạch ra theo 4 bước thực hiện [12]:

Bước 1- Kiểm thử tác vụ;

Bước 2- Kiểm thử ứng xử;

Bước 3- Kiểm thử liên tác;

Bước 4- Kiểm thử toàn hệ thống.

8.1.3.2. Kiểm thử Alpha và Beta

Kiểm thử alpha *(alpha testing)* và kiểm thử beta *(beta testing)* đều do người dùng thực hiện và đều được thực hiện trong môi trường thực.

Người phát triển phần mềm thực tế không thể nào lường hết được khách hàng sẽ dùng chương trình như thế nào. Các hướng dẫn sử dụng có thể bị hiểu sai, việc tổ hợp dữ liệu lạ lại có thể hay được dùng, cái ra dường như rõ ràng với người kiểm thử lại không dễ học cho người dùng tại hiện trường.

Kiểm thử Alpha được khách hàng tiến hành tại cơ quan của người phát triển. Phần mềm được dùng theo sự sắp đặt tự nhiên với người phát triển "nhìn qua vai" người dùng và ghi lại các lỗi khi sử dụng. Kiểm thử Alpha còn có tên khác là kiểm thử "sau lưng" và được tiến hành trong một môi trường có kiểm soát (theo kế hoạch của người phát triển).

Dữ liệu cho kiểm thử Alpha thường là dữ liệu mô phỏng.

Kiểm thử Beta được người dùng cuối tiến hành tại một hay nhiều cơ quan khách hàng. Không giống kiểm thử Alpha, người phát triển thường không có mặt. Do đó, kiểm thử Beta là việc áp dụng "sống" của phần mềm trong một môi trường mà người phát triển không thể nào kiểm soát được. Khách hàng ghi lại tất cả các vấn đề (thực hay tưởng tượng) gặp phải trong khi kiểm thử Beta và báo cáo lại những vấn đề đó cho người phát triển trong những khoảng thời gian đều đặn. Xem như một kết quả của vấn đề được báo cáo trong khi kiểm thử beta, người phát triển tiến hành sửa đổi rồi chuẩn bị đưa ra sản phẩm phần mềm cho toàn bộ khách hàng.

8.1.3.3. Kiểm thử so sánh

Các nhà nghiên cứu đã gợi ý rằng *các bản phần mềm độc lập được phát triển*. Những bản độc lập này tạo nền cho kỹ thuật kiểm thử hộp đen được gọi là *kiểm thử so sánh (comparision testing)* hay *kiểm thử dựa vào nhau (back-to-back testing).*

Khi tạo ra nhiều cài đặt cho cùng một đặc tả, các ca kiểm thử được thiết kế để dùng những kỹ thuật hộp đen khác (như phân hoạch tương đương) được xem như đầu vào của từng bản phần mềm. *Nếu kết quả ra của mỗi bản là giống nhau thì người ta giả thiết rằng tất cả các cài đặt đều đúng*. Nếu các kết quả ra là khác nhau thì từng ứng dụng sẽ được nghiên cứu lại để xác định liệu một khiếm khuyết trong một hay nhiều bản có phải là nguyên nhân sinh ra sự khác biệt hay không. Trong phần lớn các trường hợp, việc so sánh các kết quả ra có thể được thực hiện tự động.

Việc kiểm thử so sánh không đơn giản. Nếu đặc tả mà từ đó tất cả các phiên bản đã được xây dựng ra bị lỗi thì mọi bản đều có thể phản ánh lỗi đó. Mặt khác, nếu từng bản độc lập lại tạo ra kết quả giống nhau, nhưng không đúng, thì việc kiểm thử điều kiện sẽ không phát hiện được lỗi.

8.2. Các mức kiểm thử

Trong tiến trình kiểm thử, người ta chia ra thành nhiều mức kiểm thử. Về phương diện thực hành, người ta thường chia thành 4 mức cơ bản:

1- Kiểm thử đơn vị *(Unit testing);*

2- Kiểm thử tích hợp *(Integration testing);*

3- Kiểm thử hệ thống *(System testing),* bao gồm:

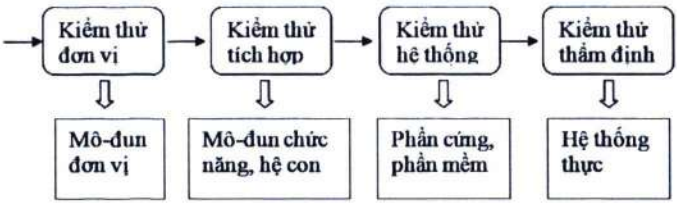
- Kiểm thử chức năng *(functional test: system and interface)*
- Kiểm thử phục hồi *(recovery test)*
- Kiểm thử chịu tải *(extra: stress and load test)*
- Kiểm thử thi hành *(performance test)*
- Kiểm thử an ninh *(security test);*

4- Kiểm thử chấp nhận *(aceptance testing)*/ thẩm định

Có 2 tiến trình thực hiện kiểm thử:

1-Tiến trình thực hiện kiểm thử tương ứng với tiến trình phát triển (theo từng mô hình).

2-Tiến trình kiểm thử thường theo mô hình *chiến lược kiểm thử tổng thể*:



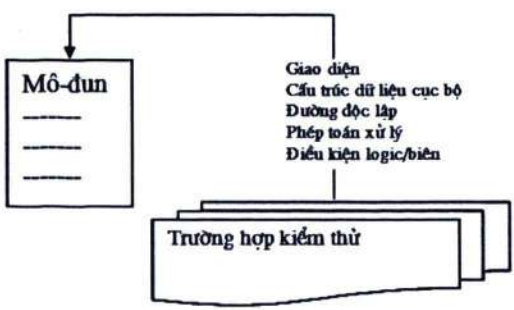
Hình 8.2. Tiến trình kiểm thử

8.2.1. Kiểm thử đơn vị

8.2.1.1. Nội dung kiểm thử đơn vị

Kiểm thử đơn vị tập trung thẩm định đơn vị nhỏ nhất của phần mềm là mô-đun. Bằng việc dùng mô tả thiết kế chi tiết làm hướng dẫn, các đường điều khiển quan trọng được kiểm thử để phát hiện ra lỗi tại biên của mô-đun. Việc kiểm thử đơn vị bao giờ cũng hướng theo "hộp trắng" và bước này có thể được tiến hành cho nhiều mô-đun.

Trong quá trình kiểm thử, cần chú ý đến hai đặc trưng cụ thể là số lượng và định dạng.



Hình 8.3. Cách tiếp cận kiểm thử đơn vị

Giao diện được kiểm thử để bảo đảm đúng thông tin vào và ra khỏi chương trình đang kiểm thử. Cấu trúc dữ liệu cục bộ được xem xét để bảo đảm rằng dữ liệu được lưu giữ tạm thời vẫn duy trì được tính toàn vẹn của nó trong tất cả các bước khi thực hiện thuật toán. Các điều kiện biên được kiểm thử để bảo đảm rằng mô-đun vận hành đúng tại các biên được thiết lập trong giới hạn xử lý. Tất cả các đường độc lập (đường cơ sở) đi qua cấu trúc điều khiển đều được cho chạy qua để bảo đảm rằng tất cả các câu lệnh trong một mô-đun đều đã được thực hiện qua đó ít nhất một lần. Cuối cùng, tất cả các đường giải quyết lỗi cũng được kiểm thử.

8.2.1.2. Kiểm thử giao diện

Việc kiểm thử luồng dữ liệu đi qua giao diện là cần thiết trước khi bất kỳ kiểm thử nào khác được khởi đầu. Nếu dữ liệu không đi vào và đi ra đúng thì tất cả các kiểm thử khác đều phải xem lại. Myer đề nghị một danh sách câu hỏi cho kiểm thử đơn vị [12].

8.2.1.3. Kiểm thử cấu trúc dữ liệu cục bộ

Cấu trúc dữ liệu cục bộ cho một mô-đun thường là nguồn gây lỗi. Các trường hợp kiểm thử nên được thiết kế để phát hiện lỗi trong các phạm trù sau:

1- Đặt kiểu không đúng hay không nhất quán;

2- Khởi đầu sai hay giá trị ngầm định;

3- Tên biến không đúng (sai chính tả hay bị cắt cụt);

4- Kiểu dữ liệu không nhất quán;

5- Giá trị ngoài miền cho phép. Gán sai địa chỉ.

Bên cạnh các cấu trúc dữ liệu cục bộ, tác động của dữ liệu toàn cục trên một mô-đun nên được xác định (nếu có thể).

8.2.1.4. Kiểm thử các đường độc lập/ đường cơ sở

Việc kiểm thử có lựa chọn các đường thực hiện là một nhiệm vị chủ chốt trong kiểm thử đơn vị. Các trường hợp kiểm thử nên được thiết kế để phát hiện ra lỗi do tính toán sai hay so sánh sai, hay luồng điều khiển không đúng. Kiểm thử đường cơ sở và chu trình là những kỹ thuật hiệu quả để phát hiện các lỗi trong một phạm vi rộng.

8.2.1.5. Kiểm thử các xử lý

Trong số nhiều lỗi chung trong tính toán cần quan tâm nhất đến các lỗi sau:

1- Hiểu nhầm hay số ưu tiên phép toán số học không đúng;

2- Phép toán trộn lẫn nhiều mốt;

3- Khởi đầu không đúng;

4- Độ chính xác không thích hợp;

5- Biểu diễn ký hiệu không đúng trong biểu thức.

Kiểm thử dòng điều khiển cần chú ý đến các sai sót về biến lặp, số vòng lặp như:

- Vòng lặp không kết thúc hoặc kết thúc không chính xác;

- Sự lặp phân kỳ khó thoát được;

- Các biến lặp bị cải biên không chính xác.

Khi kiểm thử xử lý nên chú ý đến các lỗi như:

1- Không sánh các kiểu dữ liệu khác nhau;

2- Các toán tử logic hay số ưu tiên không đúng;

3- Trông đợi sự bằng nhau trong khi lỗi độ chính xác làm cho việc bằng không thể có;

4- So sánh hay biến không đúng;

5- Việc kết thúc chu trình không đúng hay chu trình không tồn tại;

6- Lỗi đi ra khi gặp phải việc lặp khác nhau;

7- Các biến chu trình bị sửa không đúng.

8.2.1.6. Kiểm thử điều kiện logic/biên

Khi kiểm thử các điều kiện lôgic, cần chú trọng đến các sai sót về kiểu, toán tử, ngữ nghĩa:

- So sánh các kiểu dữ liệu khác nhau;

- Ưu tiên hoặc toán tử logic không đúng đắn;

- Dự đoán một biểu thức so sánh, trong khi sai số làm cho đẳng thức không chắc có thực;

- Các giá trị so sánh không đúng đắn.

Kiểm thử biên là nhiệm vụ cuối (một số người cho là quan trọng nhất) của bước kiểm thử đơn vị. Phần mềm thường bị hỏng trên biên của nó. Lỗi thường xuất hiện khi phần tử thứ n của mảng n chiều được xử lý; khi lần lặp thứ i của chu trình với i bước được gọi; khi các giá trị tối thiểu hay tối đa cho phép được gặp phải. Các trường hợp kiểm thử về cấu trúc dữ liệu, luồng điều kiện và giá trị dữ liệu ngay dưới, tại chỗ, và ngay trên cực đại và cực tiểu thường phát hiện ra lỗi.

Các sai tiềm ẩn cần được đánh giá gồm:

- Mô tả sai (khó hiểu, không đánh giá được sai hay đúng, không biết được sai đến đâu);

- Dữ liệu ghi không tương ứng với sai đã gặp;

- Điều kiện sai có trước khi xử lý sai;

- Xử lý điều kiện ngoại lệ là không đúng.

8.2.1.7. Thủ tục kiểm thử đơn vị

Sau khi mã chương trình được xây dựng, xét duyệt và thẩm định cú pháp thì việc thiết kế ca kiểm thử đơn vị bắt đầu. Mỗi phép kiểm thử thường đi kèm với một tập kết quả dự kiến.

Vì một mô-đun không phải là một chương trình độc lập nên cần phải xây dựng "*bộ lái*" (driver) /và/hoặc "*cuống*" (stub) cho phép kiểm thử đơn vị.

Trong phần lớn các ứng dụng, một bộ lái (còn được gọi là khiển trình) không gì khác hơn là một "chương trình chính" chấp nhận dữ liệu một ca kiểm thử, truyền dữ liệu đó cho mô-đun (được kiểm thử) và in ra kết quả liên quan. Cuống dùng để thay thế các mô-đun phụ thuộc (được gọi bởi) mô-đun đang được kiểm thử. Cuống dùng giao diện của các mô-đun phụ thuộc, có thể làm giảm tối thiểu các thao tác dữ liệu, in ra việc kiểm chứng khi vào và trở về.

Bộ lái và cuống biểu thị cho tổng phí. Tức là cả hai đều là đoạn chương trình cần phải viết ra nhưng không được bàn giao trong sản phẩm cuối cùng.

Việc kiểm thử đầy đủ có thể được hoãn cho tới bước kiểm thử tích hợp nếu phí dùng cho viết bộ lái và cuống quá lớn so với khả năng.

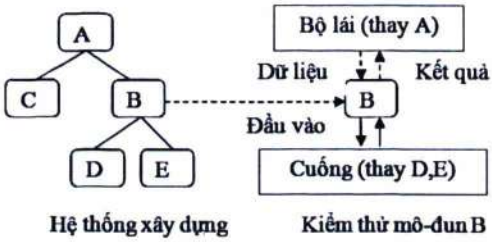
Việc kiểm thử đơn vị được đơn giản hoá khi một mô-đun với độ cố kết cao đã được thiết kế sẵn. Khi *mỗi mô-đun chỉ giải quyết cho một chức năng* thì số các trường hợp kiểm thử sẽ được giảm đi và lỗi có thể dự kiến và phát hiện dễ dàng hơn.

8.2.1.8. Kỹ thuật kiểm thử đơn vị

Như trên đã nêu, cần phát triển các *bộ lái* /và/hoặc *cuống* để kiểm thử từng đơn vị.

Bộ lái/mô-đun điều khiển là một hàm "main" điều khiển việc đưa dữ liệu vào và nhận kết quả ra của mô-đun.

Cuống dùng để thay cho các mô-đun là thứ cấp của nó.



Hình 8.4. Kỹ thuật kiểm thử đơn vị

8.2.2. Kiểm thử tích hợp

Kiểm thử tích hợp (integration testing) *là một kỹ thuật hệ thống để xây dựng cấu trúc chương trình khi tiến hành đồng thời các kiểm thử nhằm phát hiện lỗi liên kết với việc giao tiếp. Mục đích là từ các mô-đun đã kiểm thử, xây dựng nên một cấu trúc chương trình theo thiết kế.*

Kiểm thử tích hợp nhằm nhận được một bộ phận chức năng hay một hệ con tốt. Đây là một kỹ thuật có tính hệ thống để xây dựng cấu trúc chương trình. Từ các mô-đun đã kiểm thử đơn vị, xây dựng cấu trúc chương trình để bảo đảm tuân theo thiết kế.

Kiểm thử tích hợp được thực hiện dựa trên các tiêu chuẩn sau:

- *Tích hợp giao diện*: các giao diện trong và ngoài được kiểm thử mỗi khi một mô-đun (hoặc cụm) được tích hợp vào trong cấu trúc.

- *Thẩm định chức năng*: tiến hành kiểm thử chức năng đã thiết kế nhằm bộc lộ các chức năng sai.

- *Nội dung thông tin*: tiến hành kiểm thử dữ liệu đã thiết kế nhằm bộc lộ các lỗi về cấu trúc dữ liệu cục bộ và toàn cục

- *Thực thi*: tiến hành kiểm thử theo thiết kế nhằm kiểm nghiệm thực thi các hạn chế dự kiến trong quá trình thiết kế phần mềm.

Có hai cách tích hợp:

1- Tích hợp dần: *từ trên xuống, dưới lên, kẹp.*

2- Tích hợp đồng thời một lúc: *"big bang"*.

Các sai lệch có thể gặp khi tích hợp:

- Dữ liệu bị mất khi đi qua một giao diện.

- Hiệu ứng bất lợi một mô-đun vô tình gây ra cho các mô-đun khác.

- Sự kết hợp các chức năng phụ có thể không sinh ra chức năng chính mong muốn.

- Sự phóng đại các sai sót riêng rẽ có thể dẫn đến mức không chấp nhận được.

Vấn đề của cấu trúc dữ liệu toàn cục có thể để lộ ra.

Chiến lược "big bang" bao gồm các hoạt động:

- Kết hợp tất cả các mô-đun đã kiểm thử đơn vị thành chương trình;

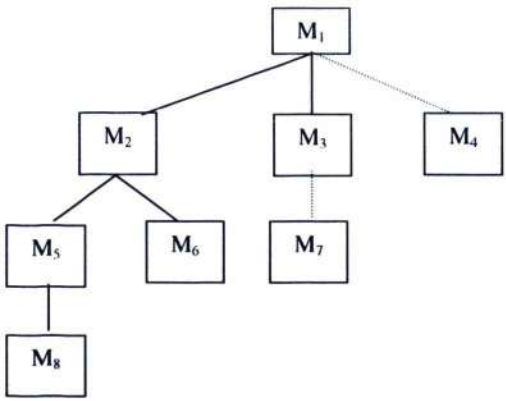
- Tiến hành kiểm thử toàn bộ chương trình.

Kết quả thực hiện chiến lược này có khả năng là một mớ hỗn độn. Với một tập các sai sót, chỉnh sửa chúng là khó khăn vì việc cô lập các nguyên nhân là rất khó, dẫn đến hiện tượng sai này được sửa, có thể xuất hiện nhiều sai khác, cứ thế triền miên!.

8.2.2.1. Chiến lược tích hợp từ trên xuống

Tích hợp từ trên xuống là cách tiếp cận tăng dần tới việc xây dựng chương trình. Các mô-đun được tích hợp bằng cách đi dần xuống qua cấp bậc điều khiển, bắt đầu với mô-đun điều khiển chính (chương trình chính).

Các mô-đun phụ thuộc (và phụ thuộc cuối cùng) vào mô-đun điều khiển chính sẽ được tổ hợp dần vào trong cấu trúc theo hoặc chiều sâu trước hay chiều rộng trước.



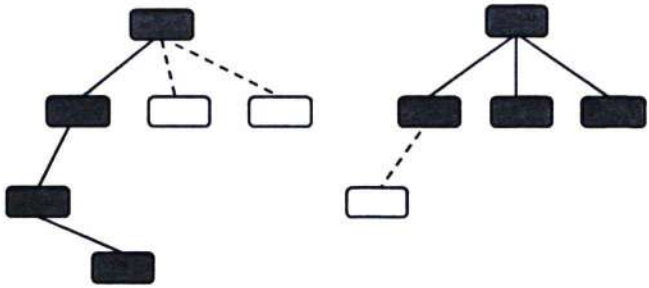
Hình 8.5. Chiến lược tích hợp từ trên xuống

Việc tích hợp chiều sâu trước sẽ tích hợp tất cả các mô-đun trên đường điều khiển chính của cấu trúc. Việc chọn một đường chính phụ thuộc vào các đặc trưng của ứng dụng.

Cách tích hợp từ trên xuống rất tiện lợi để xây dựng và kiểm soát cấu trúc chương trình. Chúng ta gộp dần các mô-đun từ trên xuống theo trật tự dòng điều khiển, bắt đầu từ mô-đun điều khiển "main", gắn từng mô-đun phụ trợ vào mô-đun điều khiển thượng cấp. Có thể theo 2 cách [12]:

1- Theo chiều "sâu" trước;

2- Theo chiều "rộng" trước.



Hình 8.6.Tiến trình tích hợp từ trên xuống

Tích hợp từ trên xuống thực hiện theo tiến trình gồm 5 bước:

1- Mô-đun điều khiển chính được dùng như bộ lái kiểm thử (test driver) *và tất cả các mô-đun phụ trợ trực tiếp được thay thế bởi các* cuống (stub).

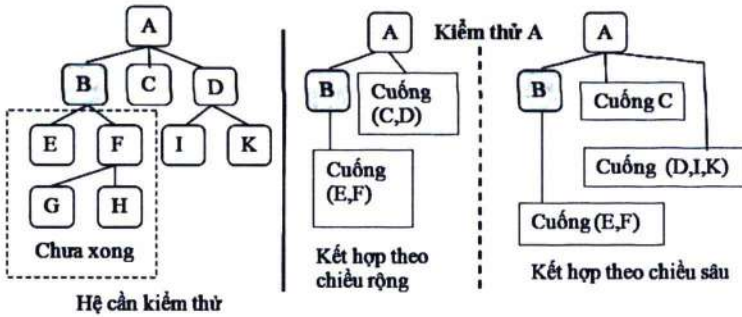
2- Thay thế dần từng cuống bằng mô-đun thực thi tương ứng.

3- Sau khi tích hợp mô-đun đó, tiến hành các kiểm thử tương ứng.

4- Khi hoàn thành các kiểm thử này thì thay một cuống khác bằng mô-đun thực (nghĩa là quay lại bước 2).

5- Có thể kiểm thử lại (toàn bộ hoặc một phần các kiểm thử trước - kiểm thử hồi quy*) để bảo đảm rằng không có sai mới nào đã sinh ra.*

Tiến trình này tiếp tục lặp lại từ bước 2 cho tới khi toàn bộ cấu trúc chương trình được xây dựng.



Hình 8.7.Sơ đồ kiểm thử tích hợp từ trên xuống

Người kiểm thử có 2 cách chọn giải pháp tích hợp từ trên xuống:

1- Trì hoãn nhiều kiểm thử cho tới khi các cuống được thay thế bằng các mô-đun thực sự.

2- Phát triển các cuống sao cho chúng hoàn thành như một phần hạn chế các chức năng mà mô-đun dưới thực sự thực hiện.

8.2.2.2. Chiến lược tích hợp từ dưới lên

Chúng ta bắt đầu xây dựng và kiểm thử từ các mô-đun nguyên tử (tức là các mô-đun ở mức thấp nhất trong cấu trúc chương trình). Việc xử lý nếu cần các mô-đun phụ trợ thì chúng thực sự đã sẵn sàng (cuống và bộ lái).

Việc sử dụng linh hoạt cả bộ lái và cuống làm dễ cho việc kiểm thử ngay cả khi chưa đủ các mô-đun mức thấp.

Khó khăn khi tích hợp chính là kiểm thử lại khi loại bỏ bộ cuống hay bộ lái sao cho hiệu quả.

Chiến lược tích hợp dưới lên có thể được thực hiện qua những bước sau:

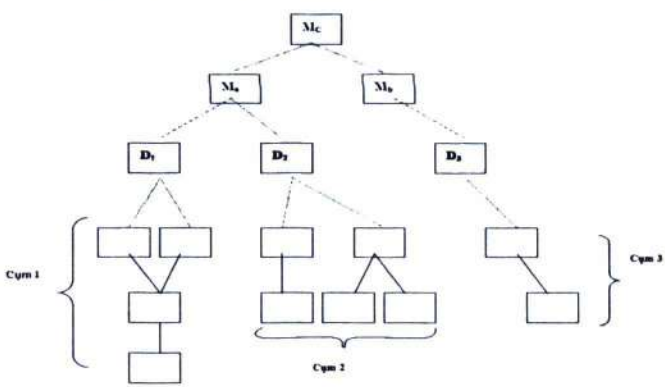
1- Các mô-đun mức thấp được tổ hợp vào trong các *cụm (cluster)* thực hiện một chức năng phụ trợ đặc biệt.

2- Một bộ lái được viết để phối hợp đầu vào và đầu ra của ca kiểm thử.

3- Kiểm thử cụm đó.

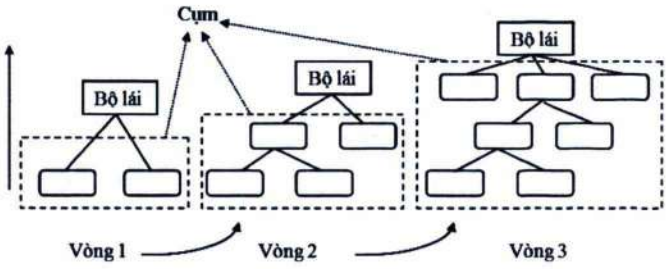
4- Tháo bỏ các bộ lái và các cụm tổ hợp ngược lên trong cấu trúc chương trình.

Việc tích hợp từ dưới lên thể hiện trong mẫu hình 8.8:



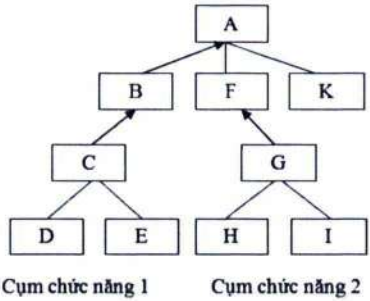
Hình 8.8. Sơ đồ tích hợp từ dưới lên

Khi việc tích hợp lên trên, nhu cầu về các bộ lái kiểm thử tách biệt ít dần. Trong thực tế, nếu hai mức đỉnh của cấu trúc chương trình được tích hợp theo kiểu trên xuống thì số các bộ lái có thể được giảm bớt khá nhiều và việc tích hợp các cụm được đơn giản rất nhiều.

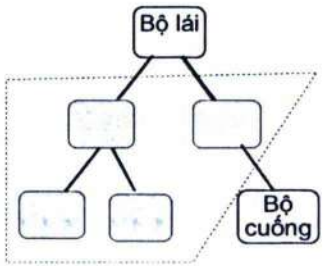


Hình 8.9. Sơ đồ kiểm thử tích hợp từ dưới lên

Các kỹ thuật tích hợp từ dưới lên có thể minh họa trên hình 8.10:



Hình 8.10. Kỹ thuật tổ chức theo nhiều cụm



Hình 8.11. Kỹ thuật sử dụng bộ lái và cuống

8.2.2.3. Phương pháp hỗn hợp (kiểm thử sandwich)

Phương pháp hỗn hợp có thể có triển vọng nhất: trên-xuống cho mức cao của cấu trúc chương trình và dưới-lên cho mức thấp (phụ trợ).

Người kiểm thử cần định rõ các mô-đun tới hạn, tức là các mô-đun có các đặc tính sau:

- Chú trọng đến một vài yêu cầu phần mềm;
- Có mức điều khiển cao (trong cấu trúc chương trình);
- Phức tạp hoặc dễ có sai (có thể dùng độ phức tạp vòng lặp, chẳng hạn);
- Có các yêu cầu thực hiện được xác định rõ.

8.2.3. Kiểm thử hệ thống

Hệ thống dựa trên máy tính gồm toàn bộ hệ thống phần mềm và môi trường vận hành (phần cứng, phần mềm nền, các hệ thống liên quan) của nó.

Kiểm thử hệ thống chỉ bắt đầu khi hệ thống có đầy đủ cả phần mềm và môi trường vận hành.

Việc kiểm thử hệ thống dễ có nguy cơ "*tắc nghẽn*" do không đồng bộ giữa các phần và các bên tham gia có thể "*đổ lỗi cho nhau*".

Việc kiểm thử hệ thống thực tế là một loại các bước kiểm thử khác nhau có mục đích chính là thử đầy đủ hệ thống dựa trên máy tính. Mặc dầu mỗi kiểm thử đều có một mục tiêu khác nhau, tất cả công việc nên kiểm chứng lại rằng mọi phần tử hệ thống đều đã được tích hợp đúng đắn và thực hiện các chức năng được cấp phát.

Các hoạt động kiểm thử hệ thống bao gồm:

1-Mô phỏng các dữ liệu xấu và các sai tiềm ẩn tại giao diện phần mềm;

2-Kiểm thử kết quả của mỗi đường liên kết;

3-Báo cáo các kết quả kiểm thử từng phần, từng loại làm chứng cứ phòng ngừa đổ lỗi cho nhau;

4-Hoạch định và thiết kế các ca kiểm thử hệ thống theo những cách nhìn khác nhau bảo đảm *phần mềm được kiểm thử đầy đủ, chính xác tất cả các loại yêu cầu.*

Dữ liệu qua giao diện có thể: sai, gây sai, phóng đại sai.

8.2.3.1. Kiểm thử chức năng (mức hệ thống)

Chức năng mức hệ thống gồm các chức năng giao diện, các chức năng mức người dùng và đầu ra cuối cùng khỏi hệ thống.

Các chức năng này thường mang tính tích hợp. Khi phát hiện sai phải quay lại kiểm thử từng phần cấu thành trước đó.

Các giao diện (người dùng, hệ thống) được xem như điểm phân định giữa các phần để kiểm thử.

8.2.3.2. Kiểm thử phục hồi (chịu lỗi)

Nhiều hệ thống dựa trên máy tính phải phục hồi các lỗi và tiếp tục việc xử lý trong một thời gian xác định trước. *Trong một số trường hợp, một hệ thống phải được phép dung sai, tức là lỗi xử lý phải không gây ra chức năng toàn bộ hệ thống dừng lại.* Trong những trường hợp khác, sai hỏng hệ thống phải được sửa chữa trong một thời kỳ xác định nếu không thiệt hại kinh tế nghiêm trọng sẽ xuất hiện.

Kiểm thử phục hồi là kiểm thử hệ thống bắt buộc phần mềm phải hỏng theo nhiều cách và kiểm thử rằng việc phục hồi được thực hiện đúng. Nếu việc phục hồi là tự động (được thực hiện bởi bản thân hệ thống) thì việc khởi đầu lại, cơ chế điểm kiểm tra, phục hồi dữ liệu và cho chạy lại sẽ được đánh giá về tính đúng đắn. Nếu việc phục hồi đòi hỏi sự can thiệp của con người thì thời gian trung bình để sửa chữa sẽ được ước lượng để xác định xem liệu nó có trong giới hạn chấp nhận được hay không.

Nhiều hệ thống cần phải phục hồi sau lỗi, để tiếp tục xử lý trong một thời gian đã đặc tả trước, có thể:

- Hệ thống cần thứ lỗi: nghĩa là xử lý lỗi không làm ngừng hoạt động của toàn hệ thống;

- Lỗi phải được khắc phục dần theo chu kỳ đặc tả.

Kiểm thử phục hồi là bắt phần mềm phải thất bại để xem khả năng phục hồi của nó đến đâu. Độ tin cậy là một độ đo đánh giá khả năng phục hồi.

8.2.3.3. Kiểm thử an ninh (sức chịu tấn công)

Bất kỳ hệ thống dựa trên máy tính mà quản lý những thông tin nhạy cảm hay gây ra những hành động mà có thể gây hại (hay gây lợi) không đúng cho các cá nhân thì cũng đều là mục tiêu cho việc xâm nhập không đúng hay bất hợp pháp.

Kiểm thử an ninh cố gắng kiểm thử rằng các cơ chế bảo vệ được xây dựng bên trong hệ thống trong thực tế sẽ bảo vệ cho hệ thống khỏi sự xâm nhập không đúng. Beizer: "An toàn của hệ thống tất nhiên phải được kiểm thử về tính mỏng manh đối với các công kích từ phía trước- nhưng cũng phải được kiểm thử về tính mỏng manh đối với công kích bên sườn hay sau lưng".

Trong khi kiểm thử an ninh, người kiểm thử đóng vai trò của cá nhân muốn thâm nhập vào hệ thống. Mọi thứ đều có thể xảy ra. Người kiểm thử có thể định lấy mật hiệu qua các phương tiện thư ký bên ngoài, có thể công kích vào hệ thống bằng những phần mềm làm riêng chuyên dụng cho việc phá vỡ bất kỳ sự phòng vệ nào đã được xây dựng, có thể tràn ngập hệ thống, do đó hệ thống từ chối việc phục vụ người khác, có thể gây các lỗi hệ thống có chủ ý, hy vọng thâm nhập vào khi đang phục hồi dữ liệu, có thể duyệt qua dữ liệu không an toàn, hy vọng tìm ra khoá để vào hệ thống.

Với đủ thời gian và tài nguyên, việc kiểm thử an ninh tốt cuối cùng sẽ thâm nhập được vào hệ thống, vai trò của người thiết kế hệ thống là làm cho việc thâm nhập này tốn kém hơn là giá trị của thông tin sẽ thu được.

Kiểm tra mọi cơ chế bảo vệ được xây dựng trong hệ thống xem có đạt hiệu quả đề ra trước các đột nhập hay không?. Xét tất cả các loại đột nhập có thể "trước mặt", "ngang sườn" và "sau lưng".

Khi thử nghiệm an ninh, người kiểm thử sẽ đóng vai trò của kẻ đột nhập.

Về nguyên tắc bảo đảm an ninh, mọi đột nhập là có thể nếu đủ thời gian và nguồn lực.

8.2.3.4. Kiểm thử chịu tải (qui mô, nhịp độ, giá trị nhạy cảm)

Trong kiểm thử chịu tải *(stress testing)*, các kỹ thuật hộp trắng và hộp đen được dùng để đánh giá chức năng và sự thi hành của chương trình ở mức bình thường.

Kiểm thử chịu tải là vận hành hệ thống khi sử dụng nguồn lực với quy mô, tần suất, cường độ dị thường. Chú ý rằng:

1- Các kiểm thử đặc biệt có thể được kỹ thuật để sinh ra 10 ngắt trong 1 giây, khi tỷ lệ trung bình chỉ là 1 hay 2;

2- Tỷ lệ dữ liệu vào có thể tăng lên theo một cấp độ lớn nào đó để xác định cách các chức năng vào sẽ đáp ứng;

3- Các trường hợp kiểm thử đòi hỏi bộ nhớ tối đa hay các tài nguyên khác có thể phải thực hiện ;

4- Các trường hợp kiểm thử có thể làm hỏng hệ điều hành thực được thiết kế;

5- Các trường hợp kiểm thử có thể gây ra việc tiêu tốn dữ liệu thường trú trên đĩa cũng cần phải được thiết kế ra.

Một loại khác của kiểm thử áp lực là kiểm thử độ nhạy: cố gắng làm bộc lộ các tổ hợp dữ liệu (lớp dữ liệu có hiệu lực) hay các sự kiện mà có thể gây ra việc xử lý không ổn định hoặc không chính xác.

8.2.3.5. Kiểm thử thi hành

Kiểm thử thi hành (Performance testing) bảo đảm *thông suốt, kịp thời, hiệu quả*. Với hệ nhúng và hệ thời gian thực, phần mềm cung cấp chức năng nhưng không phù hợp với các yêu cầu thi hành đều là không chấp nhận được.

Kiểm thử thi hành được thiết kế để kiểm thử việc vận hành (run-time) của phần mềm khi hệ thống được tích hợp. Kiểm thử thi hành xuất hiện trong tất cả các bước của quá trình kiểm thử, tuy nhiên chỉ khi tất cả các phần tử của hệ thống đã được tích hợp thì kiểm thử mới thực sự là chắc chắn, tincậy.

Việc thi hành đúng bao gồm cả mặt số lượng, chất lượng (hoạt động, hiệu năng, ràng buộc).

Kiểm thử thi hành thường gắn với kiểm thử áp lực, vì cả hai đều đòi hỏi các công cụ phần cứng và phần mềm chuyên dụng: Cần đo sự tổnghợp nguồn lực (trong, ngoài) và nhờ thiết bị ngoại lai để giám sát các khoản vận hành, các sự kiện ngắt (log) khi nó xuất hiện, có thể lấy mẫu các trạng thái này. Có

thể làm bộc lộ các tình thế dẫn đến sự suy giảm hiệu năng hoặc thất bại hệ thống tiềm ẩn.

8.2.3.6. Phương pháp kiểm thử hệ thống

Kiểm thử hệ thống có nhiều loại, mỗi loại có cách nhìn khác nhau nên có phương pháp khác nhau.

Phương pháp chung để kiểm thử hệ thống là phương pháp mô hình hóa. Mỗi loại phương pháp sử dụng các mô hình khác nhau. Có thể phải kết hợp hoặc dùng nhiều mô hình cho một loại kiểm thử. Song việc kiểm thử hiệu quả tùy thuộc khả năng và kinh nghiệm của người tiến hành [12].

8.2.4. Kiểm thử chấp nhận/hợp lệ

Kết thúc kiểm thử tích hợp, phần mềm được lắp ráp trong 1 gói, các kiểm thử cuối cùng bắt đầu - kiểm thử chấp nhận hay hợp lệ *(acceptation/validation testing)*.

Hợp lệ là thắng lợi nếu các chức năng phần mềm ở một chừng mức nào đó thoả mãn được sự mong đợi hợp lý của người đặt hàng. Mục tiêu hợp lệ nhằm xem phần mềm có đáp ứng được yêu cầu khách hàng không ?

"Mong đợi hợp lý" của khách hàng thể hiện:

- Trong đặc tả yêu cầu phần mềm bao gồm cả phần mô tả được gọi là tiêu chuẩn kiểm thử phần mềm;

- Thông qua đánh giá trực tiếp khi sử dụng.

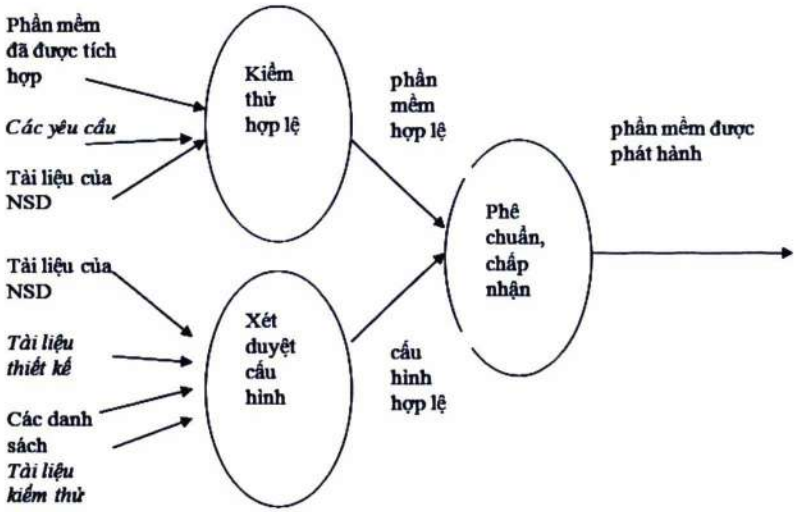
Hợp lệ phần mềm được thực hiện thông qua một loạt các kiểm thử hộp đen để thuyết minh sự phù hợp của nó với các yêu cầu.

Sau khi tiến hành mỗi trường hợp kiểm thử, một trong hai điều kiện có thể tồn tại:

1- Các đặc trưng chức năng và hiệu năng tuân thủ với đặc tả và được chấp nhận;

2- Một độ lệch so với đặc tả được phát hiện ra và một danh sách các khiếm khuyết được tạo ra.

Một phần tử quan trọng của tiến trình làm hợp lệ là xét duyệt cấu hình. Ý định của việc xét duyệt được minh hoạ trong hình dưới đây là để bảo đảm rằng tất cả các phần tử của cấu hình phần mềm đã được phát triển đúng đắn và được phân loại với chi tiết cần thiết để hỗ trợ cho giai đoạn bảo trì trong vòng đời phần mềm. Việc xét duyệt cấu hình còn được gọi là *kiểm toán.*



Hình 8.12. Quá trình phê chuẩn chấp nhận

Một kế hoạch phác họa những lớp kiểm thử cần tiến hành và một thủ tục kiểm thử xác định các ca kiểm thử sẽ thực hiện để thuyết minh sự phù hợp với các yêu cầu.

Cả kế hoạch và thủ tục được thiết kế để bảo đảm rằng:

o Tất cả các yêu cầu được thoả mãn;

o Các yêu cầu thi hành đã đúng đắn;

o Tài liệu đầy đủ, chính xác;

o Các yêu cầu khác là thoả đáng.

Sau mỗi ca kiểm thử, phần mềm ở vào một trong hai trường hợp:

1- Các đặc tính chức năng và sự thực hiện phù hợp với đặc tả và được chấp nhận;

2- Các lệch lạc so với đặc tả và mong đợi được phát hiện và một danh sách các khiếm khuyết được tạo ra.

Các sai sót không chỉnh sửa trong giai đoạn này. Thường phải thảo luận lại với khách hàng để thiết lập một phương pháp giải quyết các sai lệch này.

Khi phần mềm được làm theo yêu cầu của một khách hàng thì một loạt các kiểm thử chấp nhận sẽ được tiến hành để cho khách hàng chấp nhận. Kiểm thử chấp nhận được người dùng cuối cùng thực hiện chứ không phải là người phát triển hệ thống.

Nếu phần mềm được phát triển như một sản phẩm cho nhiều người dùng thì việc thực hiện các kiểm thử chấp nhận chính thức cho từng khách hàng là điều không thực tế. Phần lớn những người xây dựng sản phẩm phần mềm đều dùng một tiến trình gọi là kiểm thử Alpha và Beta để phát hiện lỗi mà dường như chỉ người dùng cuối cùng mới có thể tìm ra.

8.3. Phương pháp và kỹ thuật kiểm thử

Thiết kế kiểm thử cho phần mềm và các sản phẩm kỹ nghệ khác có thể có tính thách đố như việc thiết kế ban đầu cho chính bản thân sản phẩm. Người kỹ sư phần mềm thường kiểm thử phần mềm sau khi lập trình xong. Các ca kiểm thử đòi hỏi vừa đúng lại vừa đủ. Mặt khác, kiểm thử cần được thiết kế sao cho có khả năng cao nhất để tìm ra lỗi với thời gian và công sức ít nhất.

Có rất nhiều phương pháp thiết kế trường hợp kiểm thử cho phần mềm. Những phương pháp này cung cấp cho người phát triển một cách tiếp cận hệ thống tới việc kiểm thử. Quan trọng hơn, những phương pháp này đưa ra một cơ chế có thể giúp đảm bảo tính đầy đủ của kiểm thử và đưa ra khả năng cao nhất để phát hiện ra lỗi trong phần mềm.

Bất kỳ sản phẩm kỹ nghệ nào (và hầu hết mọi thứ khác) đều có thể được kiểm thử theo một trong hai cách:

1- *Kiểm thử chức năng/hộp đen: cho dữ liệu đầu vào đúng/sai, kiểm tra đầu ra đúng /sai, tức là kiểm thử xem từng chức năng có vận hành đúng không, không quan tâm đến cấu trúc bên trong của chức năng đó;*

2- *Kiểm thử cấu trúc/hộp trắng*: không những quan tâm đến mối quan hệ giữa đầu vào và đầu ra của chức năng đó mà còn quan tâm đến cấu trúc bên trong, quan tâm chi tiết đến từng đầu vào đầu ra của các thành phần cấu thành trong đó và cả sự ăn khớp giữa chúng nữa, tức là bảo đảm rằng sự vận hành bên trong thực hiện đúng theo đặc tả và tất cả các thành phần bên trong đều được quan tâm và được kiểm tra một cách chi tiết.

Đối với phần mềm máy tính, kiểm thử hộp đen biểu thị việc *kiểm thử được tiến hành tại giao diện phần mềm*. Mặc dầu chúng được thiết kế để phát hiện ra lỗi, kiểm thử hộp đen được dùng để thể hiện rằng các chức năng phần mềm đã vận hành, cái vào được chấp nhận đúng, và cái ra được tạo ra đúng, tính toàn vẹn của thông tin ngoài (như tệp dữ liệu) là được duy trì. Phép kiểm thử hộp đen xem xét một số khía cạnh của hệ thống mà ít để ý tới cấu trúc logic bên trong của phần mềm.

K*iểm thử hộp trắng được hướng tới việc xem xét kỹ về chi tiết thủ tục.* Các đường logic đi qua phần mềm được kiểm thử bằng cách đưa ra các trường hợp kiểm thử, vốn thực hiện trên một tập xác định các điều kiện và /hoặc chu trình. "Trạng thái của chương trình" có thể được xem xét tại nhiều điểm khác nhau để xác định liệu trạng thái dự kiến hay khẳng định có tương ứng với trạng thái thực tại không.

Thoáng nhìn dường như là việc kiểm thử hộp trắng rất thấu đáo sẽ dẫn tới "chương trình đúng đắn 100 phần trăm". Mọi điều ta cần làm là xác định tất cả các con đường logic, xây dựng các trường hợp kiểm thử để vét hết logic chương trình. Nhưng việc kiểm thử vét cạn lại có một số vấn đề. Ngay cả với *những chương trình nhỏ, số các đường logic có thể cũng có thể rất lớn.*

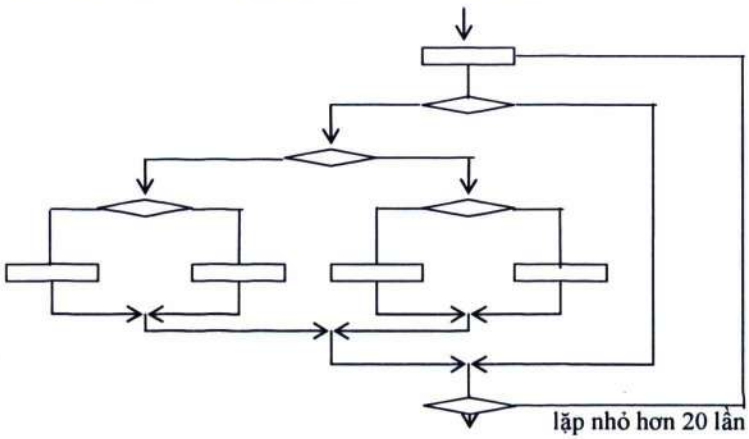
Khi xét sơ đồ khối được vẽ trong hình 8.13, thiết kế thủ tục được minh hoạ bởi sơ đồ này có thể tương ứng với một chương trình 100 dòng trong PASCAL với một chu trình có thể được thực hiện không nhiều hơn 20 lần. Có xấp xỉ 10^{14} con đường có thể có để thực hiện.

Để đặt con số này vào khung cảnh, ta giả sử rằng một bộ xử lý kiểm thử thần kỳ ("thần kỳ" vì không bộ xử lý nào như thế tồn tại cả) đã được xây dựng cho việc kiểm thử vét cạn. Bộ xử lý này có thể xây dựng ra một trường

hợp thử, thực hiện nó và đánh giá kết quả trong 1 phần mười nghìn giây. Làm việc 24 giờ một ngày, 365 ngày một năm, bộ xử lý này sẽ phải làm việc trong 3170 năm để kiểm thử chương trình được biểu thị trong hình 8.13. *Như vậy, việc kiểm thử vét cạn là không thể thực hiện được đối với các hệ thống phần mềm lớn.*

Tuy nhiên việc kiểm thử hộp trắng không nên bị bỏ qua không xét vì không thực tế. Người ta có thể chọn ra một số có giới hạn những đường logic quan trọng và thực hiện chúng.

Có thể thăm dò các cấu trúc dữ liệu quan trọng về tính hợp lệ. Các thuộc tính của cả kiểm thử hộp đen lẫn hộp trắng có thể được tổ hợp để đưa ra một cách tiếp cận để làm hợp lệ cho giao diện phần mềm và bảo đảm có chọn lựa rằng công việc bên trong của phần mềm là đúng đắn.



Hình 8.13. Các vấn đề với kiểm thử vét cạn

8.3.1. Kiểm thử hộp trắng

8.3.1.1. Bản chất kiểm thử hộp trắng

Đối tượng kiểm thử hộp trắng (white box) là mã nguồn ở các mô-đun đơn vị. Kiểm thử hộp trắng (chủ yếu là kiểm thử cấu trúc điều khiển) thể hiện một phương pháp thiết kế trường hợp kiểm thử *có dùng cấu trúc điều khiển của thiết kế thủ tục để suy ra các trường hợp kiểm thử.*

Bằng việc dùng các phương pháp kiểm thử hộp trắng, người kỹ sư phần mềm có thể suy dẫn ra các trường hợp kiểm thử mà:

1- Bảo đảm rằng mọi con *đường độc lập* bên trong một mô-đun đều được thực hiện qua ít nhất một lần;

2- Thực hiện tất cả các *quyết định logic* trên nhánh đúng và sai;

3- Thực hiện tất cả các *chu trình* tại biên giới của chúng và bên trong các giới hạn chức năng;

4- Thực hiện các *cấu trúc dữ liệu* bên trong để đảm bảo tính hợp lệ của chúng.

Bản chất của khiếm khuyết phần mềm chính là lý do đầu tiên phải kiểm thử hộp trắng:

- Các lỗi logic và các giả thiết không đúng tỷ lệ nghịch với xác suất thực hiện một đường mạch thực hiện trong chương trình. Lỗi dễ sinh ra khi chúng ta thiết kế và thực hiện các hàm, điều kiện hay điều khiển nằm ngoài luồng mạch chính. Việc nào được xử lý thường xuyên sẽ được hiểu rõ, ngược lại sẽ không hiểu rõ và thường có lỗi ẩn trong đó.

- Luồng logic của một chương trình đôi khi khác thường, nghĩa là những giả thiết vô ý thức của chúng ta về luồng điều khiển và dữ liệu có thể dẫn tới các lỗi thiết kế mà chỉ phát hiện được khi kiểm thử.

- Lỗi gõ vào là ngẫu nhiên. Khi một chương trình được dịch sang chương trình gốc trong ngôn ngữ lập trình thì có thể là một số lỗi gõ máy sẽ xuất hiện. Nhiều lỗi sẽ được cơ chế kiểm tra cú pháp phát hiện, nhưng những lỗi khác sẽ không bị phát hiện cho đến khi việc kiểm thử bắt đầu. Rất có thể là một lỗi gõ sai sẽ tồn tại trên một đường logic chưa rõ như trên một đường đi của luồng chương trình.

Mỗi một trong các lý do này đều đưa ra luận cứ để tiến hành các kiểm thử hộp trắng. Việc kiểm thử hộp đen, dù làm kỹ lưỡng đến đâu vẫn có thể sót những loại lỗi đã nêu ở trên. Theo Beizer: "Lỗi ẩn nấp trong các ngóc ngách và tập hợp tại biên giới". Việc kiểm thử hộp trắng có nhiều khả năng phát hiện ra chúng hơn.

8.3.1.2. Nội dung kiểm thử hộp trắng

Nội dung kiểm thử bao gồm các chi tiết thủ tục (xử lý, thuật toán), các con đường logic (luồng điều khiển), các trạng thái của chương trình (dữ liệu cục bộ). Cụ thể là kiểm thử:

- *Tất cả các lệnh (đầy đủ);*
- *Tất cả các điều kiện lôgic (rẽ nhánh, điều kiện);*
- *Tất cả các chu trình trong chương trình (lặp lại);*
- *Tất cả các cấu trúc dữ liệu được dùng (dữ liệu cục bộ);*
- *Tất cả các tiến trình từ đầu cho đến khi kết thúc (luồng điều khiển).*

Câu hỏi được dùng ở đây là: *Yêu cầu? Lý do? Các kỹ thuật dùng và nội dung của nó? Chiến lược sử dụng?*

8.3.1.3. Yêu cầu kiểm thử hộp trắng

Một số yêu cầu đặt ra với kiểm thử hộp trắng:

- Tất cả các con đường độc lập (*independent paths*) trong mô-đun cần được thực hiện ít nhất 1 lần;

- Tất cả các ràng buộc logic được thực hiện cả hai phía đúng (true) và phía sai (false);

- Tất cả các vòng lặp ở biên của nó và các biên vận hành phải được thực hiện;

- Tất cả các cấu trúc dữ liệu nội tại được dùng để bảo đảm hiệu lực thi hành của nó.

8.3.1.4. Lý do kiểm thử hộp trắng

Một số lý do có thể nêu ra:

- Các sai logic và giả thiết sai tỷ lệ nghịch với xác suất để 1 con đường logic được thi hành (bản chất của khiếm khuyết phần mềm đã nêu ở trên);

- Thực tế, mọi con đường lôgic đều có thể được thi hành trên một cơ sở nhất định;

- Có những sai chính tả có thể là ngẫu nhiên trên đường không được kiểm tra.

8.3.1.5. Các kỹ thuật sử dụng

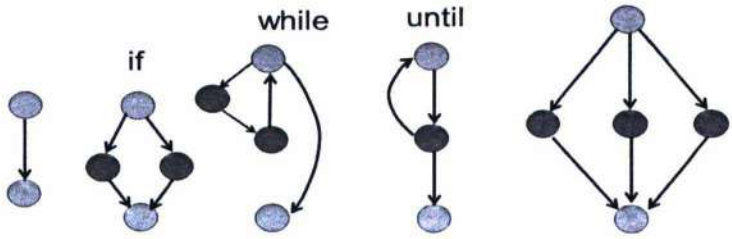
Để kiểm thử hộp trắng, người ta thường sử dụng các kỹ thuật sau:

- Đồ thị dòng (Tom McCabe đưa ra đầu tiên 1976);
- Ma trận kiểm thử (số đường đi, trọng số);
- Điều kiện lôgic với chiến lược kiểm thử miền và nhánh;
- Điều khiển theo dòng dữ liệu;
- Các cấu trúc chu trình - giá trị đặc trưng.

Các kỹ thuật kiểm thử hộp trắng đã được học trong giáo trình kỹ nghệ phần mềm cơ sở, cấp đại học [12]. Ở đây, chúng ta chỉ trình bày kỹ thuật đầu tiên - kỹ thuật đồ thị dòng, một kỹ thuật dễ dùng và khá phổ biến.

Kỹ thuật kiểm thử hộp trắng đầu tiên được Tom McCabe đề nghị là *"kiểm thử đường cơ sở"*. Phương pháp đường cơ sở giúp cho người thiết kế trường hợp kiểm thử có thể suy dẫn ra một cách đo độ phức tạp logic của thiết kế thủ tục và dùng cách đo này như một hướng dẫn để xác định một tập cơ sở các đường thực hiện. Các trường hợp kiểm thử được suy dẫn ra để thực hiện một tập cơ sở, được bảo đảm để thực hiện mọi câu lệnh trong chương trình ít nhất một lần trong khi kiểm thử.

Trong phương pháp đường cơ sở, một hệ thống *ký pháp đơn giản được dùng để biểu diễn cho luồng điều khiển, được gọi là đồ thị dòng* (hay đồ thị chương trình). Đồ thị dòng mô tả cho dòng điều khiển logic dùng ký pháp được minh hoạ trong hình 8.14. *Mỗi kết cấu có cấu trúc* đều có một ký hiệu đồ thị dòng tương ứng.



Hình 8.14. Các cấu trúc cơ bản của đồ thị dòng (sequence, if, while, until, case)

Đồ thị dòng thực chất là một kỹ thuật dựa trên cấu trúc điều khiển của chương trình. Nó gần giống đồ thị luồng điều khiển của chương trình.

Đồ thị dòng nhận được từ đồ thị luồng điều khiển bằng cách:

- Gộp các lệnh tuần tự và điều khiển liên tiếp thành một lệnh;
- Thay lệnh rẽ nhánh của các đường điều khiển bằng một nút *"vị tự"*.

Cấu trúc đồ thị dòng gồm:

- Mỗi nút (hình tròn) biểu thị 1 hay một số lệnh tuần tự và rẽ nhánh, hoặc thay cho điểm phân nhánh hay hội tụ các đường điều khiển.
- Mỗi cạnh nối hai nút biểu diễn dòng điều khiển.

Kết quả đồ thị dòng thể hiện:

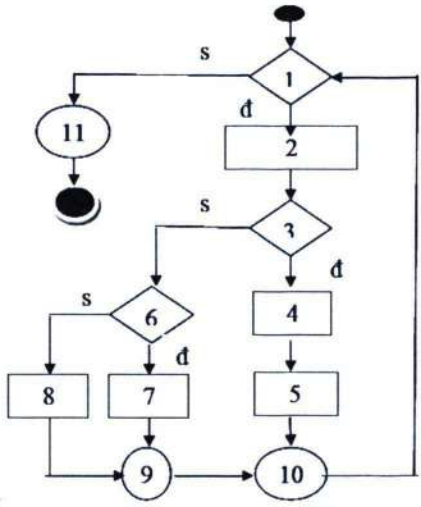
- Chia mặt phẳng thành nhiều miền.
- Có nút vị tự biểu thị sự phân nhánh của các cung.
- Mỗi cung nối từng cặp nút biểu diễn luồng điều khiển.

Ví dụ 8.1:

Xét 1 cấu trúc điều khiển chương trình đã được gán nhãn

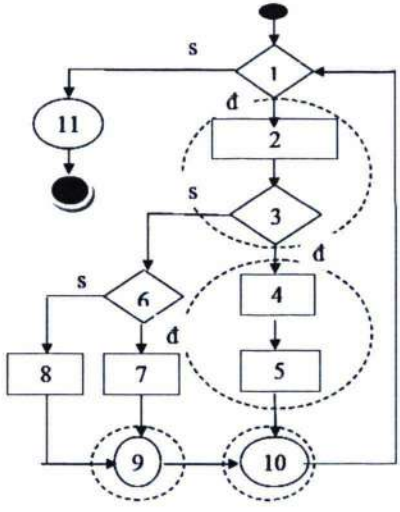
```
1  Do while records remain
2       Read record
3       If record field1 = 0
4            then process record;
                      store in buffer;
5            increment counter;
6       Else if record field2 = 0
7                 then reset record;
8                   Else process record;
                            store in file;
9            Endif
10      Endif
11 Enddo
```

Chúng ta cần vẽ sơ đồ điều khiển của chương trình tương ứng:



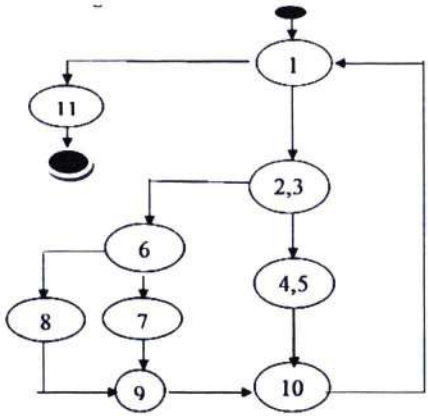
Hình 8.15.Sơ đồ điều khiển của chương trình

Tiếp tục, thu gọn sơ đồ này:



Hình 8.16. Sơ đồ thu gọn luồng điều khiển

để có được đồ thị dòng:



Hình 8.17. Đồ thị dòng

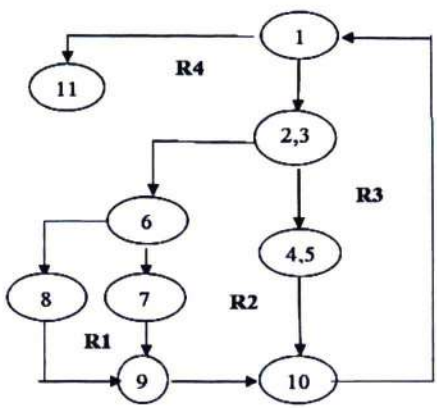
Các thông số của đồ thị dòng gồm:

9 nút (=N), trong đó:

3 nút là vị tự (= P)

11 cung (= E)

Đồ thị dòng chia mặt phẳng thành 4 miền: R1, R2, R3, R4



Hình 8.18. Độ phức tạp chu trình xác định từ đồ thị dòng

Để bảo đảm mọi câu lệnh đều được kiểm thử ít nhất 1 lần, cần tìm được tất cả các đường điều khiển độc lập trong chương trình (khác nhau ít nhất 1 lệnh).

Số các đường độc lập của một chương trình là giới hạn trên số ca kiểm thử cần tiến hành. Nó được gọi là độ phức tạp chu trình của chương trình.

Tập các đường độc lập/cơ bản (*basic paths*) của một chương trình trùng với các đường độc lập của đồ thị dòng (tìm đơn giản hơn).

Tính toán độ phức tạp chu trình:

Độ phức tạp chu trình V(G) của đồ thị G được tính theo các cách sau:

V(G) = E - N + 2 (= 11-9+2 = 4)

V(G) = số miền phẳng (= 4)

V(G) = P + 1 (= 3+1 = 4)

Trong đó: E = số cung; N = số nút; P = số nút vị tự.

Với ví dụ về đồ thị dòng ở trên ta có: V(G) = 4.

Xác định các ca kiểm thử:

Từ đồ thị dòng, xác định được độ phức tạp chu trình V(G)=4 và suy ra cần thiết kế 4 đường kiểm thử, tạo thành tập đường cơ bản trong bảng 8.1.

Bảng 8.1. Tập đường cơ bản

Tập đường cơ bản							
a	1	11					
b	1	2-3	4-5	10	1		
c	1	2-3	6	7	9	10	1
d	1	2-3	6	8	9	10	1

8.3.2. Kiểm thử hộp đen

8.3.2.1. Bản chất kiểm thử hộp đen

Kiểm thử hộp đen (black box) là kiểm thử theo yêu cầu chức năng. Đối tượng kiểm thử gồm các *mô-đun, hệ con, toàn hệ thống*

Kiểm thử hộp đen có những đặc trưng:

- Thuyết minh: các chức năng đủ và vận hành đúng;
- Thực hiện: qua giao diện;
- Cơ sở: đặc tả, điều kiện vào/ra và cấu trúc dữ liệu;
- Ít chú ý tới cấu trúc logic nội tại của nó.

8.3.2.2. Mục đích kiểm thử hộp đen

Tìm *các loại sai* liên quan:

- Chức năng: đủ, đúng đắn;
- Giao diện: vào, ra: đủ, phù hợp, đúng, tiện lợi;
- Cấu trúc, truy cập dữ liệu: thông suốt, đúng đắn;
- Thực thi: trôi chảy, kịp thời, chịu lỗi, phục hồi;
- Khởi đầu - kết thúc: thông suốt.

8.3.2.3. Các kỹ thuật kiểm thử hộp đen

8.3.2.3.1. Kỹ thuật Phân hoạch tương đương

Kỹ thuật Phân hoạch tương đương là một kỹ thuật của kiểm thử hộp đen.

Về mặt nguyên tắc, kỹ thuật này chia (phân hoạch) miền vào của chương trình thành các lớp dữ liệu để lập ca kiểm thử theo mỗi lớp đó.

Cơ sở của kỹ thuật này là dữ liệu trong một lớp tương đương tác động như nhau lên chương trình, tạo ra cùng một trạng thái: đúng hay sai của chương trình.

Mục tiêu của phương pháp phân hoạch là tìm ra một ca kiểm thử để bộc lộ một lớp sai, từ đó rút gọn số ca kiểm thử cần phát triển. Ca kiểm thử được thiết kế cho từng lớp tương đương.

Vấn đề đặt ra là chọn lớp tương đương như thế nào ?

Phương châm xác định các lớp tương đương:

1. Điều kiện vào là một vùng rộng giới hạn một miền hay những giá trị đặc biệt thì cần xác định:

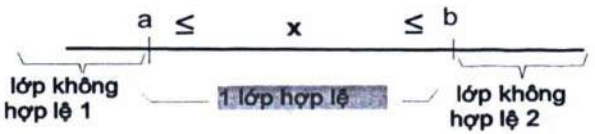
- 1 lớp tương đương hợp lệ;
- 2 lớp tương đương không hợp lệ.

2. Điều kiện vào một thành phần của một tập hoặc điều kiện Bool thì cần xác định:

- *1 lớp tương đương hợp lệ;*
- *1 lớp tương đương không hợp lệ.*

Ví dụ 8.2:

Điều kiện là 1 vùng giới hạn 1 miền giữa a và b:



Hình 8.19. Phân chia lớp tương đương [12]

Điều kiện là một phần của một tập: x nhận giá trị nguyên dương sao cho:

- 1 lớp tương đương hợp lệ: *các số nguyên dương;*
- 1 lớp tương đương không hợp lệ: *các số nguyên âm.*

8.3.2.3.2. Kỹ thuật Phân tích giá trị biên

Người ta nhận thấy rằng:

- *Các sai có xu hướng xuất hiện ở biên của vùng dữ liệu (hơn là ở "trung tâm");*
- *Các sai có thể cả trong và ngoài biên;*
- *Kiểm thử không chỉ chú ý đến các dữ liệu biên mà còn chú ý đến các dữ liệu sát biên (trong, ngoài);*
- *Các ca kiểm thử được xác định nhằm thực hiện các giá trị biên và sát biên.*

Việc chọn lớp tương đương giá trị biên theo cách phân tích giá trị biên:

1. Nếu điều kiện vào là một miền giới hạn bởi a và b thì cần thiết kế các ca kiểm thử cho cả a và b, và cả trên, dưới a và b;

2. Nếu điều kiện vào đặc tả một số giá trị thì thiết kế các ca kiểm thử cho cả các số trên và dưới số nhỏ nhất và lớn nhất.

Phạm vi áp dụng cho giá trị biên ra:

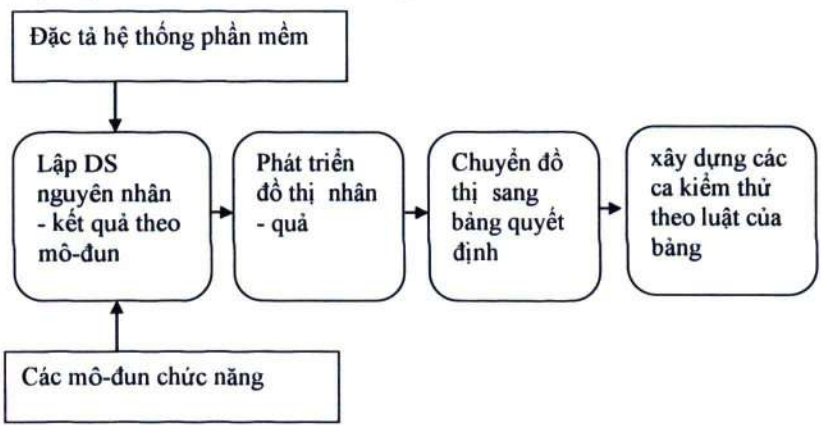
- Áp dụng cách 1 và 2 cho cả điều kiện ra.

- Áp dụng điều kiện giá trị biên cho cả chương trình trung gian có các biên của cấu trúc dữ liệu được mô tả.

8.3.2.3.3. Kỹ thuật đồ thị nhân- quả

Kỹ thuật đồ thị nhân - quả là một kỹ thuật để thiết kế ca kiểm thử. Nó cung cấp một biểu diễn chính xác giữa các điều kiện logic (đầu vào) và các hành động tương ứng (đầu ra- kết quả).

Kỹ thuật đồ thị nhân - quả được xây dựng dựa trên các mô-đun chức năng, lôgic tiến trình và đặc tả hệ thống.



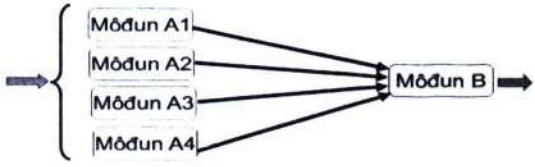
Hình 8.20. Các bước tiến hành theo kỹ thuật đồ thị nhân- quả [12]

Ví dụ 8.3:

Bảng 8.2. Danh sách nhân- quả theo mô-đun

Mô-đun	Nguyên nhân	Kết quả	Định danh nhân -quả
A	Số > a	đúng	A1
	Số ≥ a	nghi ngờ	A2
	Số = a	nghi ngờ	A3
	Số < a	sai	A4
B	Số nguyên	đúng	B

Có nhiều công cụ để xây dựng đồ thị nhân- quả. Đồ thị có hướng hay được dùng hơn cả.

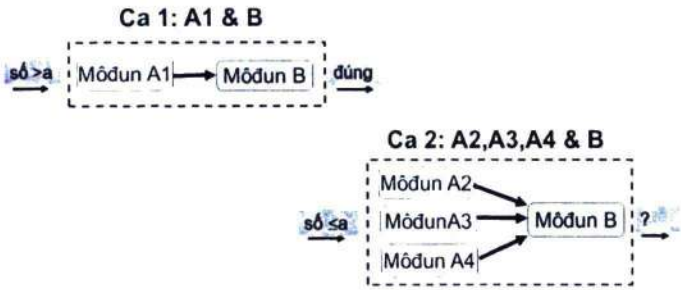


Hình 8.21.Xây dựng đồ thị nhân -quả bằng đồ thị [12]

Bảng 8.3. Bảng quyết định của đồ thị nhân -quả

Định danh	Điều kiện	Đúng	Nghi ngờ	Sai
A1	Số > a	X		
A2,A3	Số ≥ a		X	
A4	Số < a			X
B	Số nguyên	X		
...	...	..	..	..

Chọn 2 ca kiểm thử:



Hình 8.22.Các phương án lựa chọn ca kiểm thử [12]

8.3.3. Kiểm thử ứng dụng web

Kiểm thử website là một thành phần trong kiểm thử phần mềm tập trung vào các ứng dụng web. Khi kiểm thử hệ thống ứng dụng Web đều phải tuân thủ đúng theo các yêu cầu khi kiểm thử một hệ thống ứng dụng nói chung [12].

Tính phổ biến của các ứng dụng Web là hoạt động theo mô hình Client - Server hoặc môi trường Web. Việc thực hiện kiểm thử cũng khá phức tạp khi xác định nỗ lực cho kiểm thử Web. Người kiểm thử không chỉ kiểm thử trên một môi trường mà còn phải kiểm thử trên nhiều môi trường khác.

Cấu trúc Client-Server liên quan tới ba đối tượng tách rời là Server, Client, và Network. Sự kết nối giữa chúng có thể gây lên lỗi. Do đó, quá trình kiểm thử phải bao quát hết được sự thực hiện của Server và network, sự thực hiện của toàn bộ hệ thống cũng như các chức năng liên quan. Nhiều công cụ kiểm thử Web cho phép giám sát, đo lường, kiểm thử và đoán toàn bộ chương trình.

Một số đặc tính thường phải kiểm thử trong ứng dụng Web:

- Kiểm tra các chức năng chạy trên các trình duyệt khác nhau.
- Kiểm tra tải đối với cấu hình máy khác nhau.

- Kiểm tra thời gian mở một trang web thường 3-5 giây/ 1trang.

- Kiểm tra số lượng người dùng tối đa, đồng thời (Load testing).

- Kiểm tra tính bảo mật của hệ thống trên các trình duyệt.

Các loại kiểm thử phổ biến trong kiểm thử ứng dụng web:

1- *Kiểm thử giao diện*: bao gồm kiểm thử thiết kế giao diện người dùng và kiểm thử thực thi giao diện người dùng.

2- *Kiểm thử chức năng*, trong đó, người ta sử dụng các phương pháp:

- Kiểm thử đơn giản chấp nhận chức năng (FAST)

- Kiểm thử chức năng hướng tác vụ (TOFT)

- Kiểm thử lỗi ép buộc (FET)

- Kiểm thử điều kiện biên hoặc phân tích lớp tương đương.

3- Kiểm thử khả năng tương thích và cấu hình (phía trình chủ, phía trình khách, trình duyệt web).

4- Kiểm thử bảo mật (Cửa sau, Kiểm thử tên tài khoản và mật khẩu, Kiểm thử sự rò rỉ thông tin, Kiểm thử việc sử dụng GET và POST, Các tấn công chèn vào lệnh SQL, Kiểm thử lỗi tràn vùng đệm, Sự tin cậy vào các script phía trình khách).

5- Kiểm thử hiệu năng, trong đó, người ta sử dụng các độ đo hiệu năng sau:

- Số giao tác trên mỗi giây;

- Số yêu cầu trên mỗi giây;

- Kết nối đồng thời;

- Thông lượng;

- Thời gian kết nối;

- Thời gian gửi;

- Thời gian nhận;

- Thời gian xử lý;

- Thời gian giao tác;

- Số giao tác thất bại trên mỗi giây;

- Số yêu cầu trên mỗi giây;

- Số yêu cầu thất bại trên mỗi giây.

Ngày nay, việc kiểm thử web chủ yếu tập trung vào kiểm thử chức năng để kiểm thử các thành phần web để đảm bảo thực hiện đúng chức năng như mong muốn. Kiểu kiểm thử này thường bao gồm sự phân tích các thành phần web và kiểm tra sự phù hợp đối với các tiêu chuẩn và các đặc tả mở rộng.

Các dạng kiểm thử chức năng thông dụng:

- Kiểm thử cú pháp HTML: Kiểm tra các file HTML để xác định sự phù hợp với những đặc tả ngôn ngữ và các chuẩn tài liệu;

- Kiểm thử các liên kết (links checking): Kiểm tra toàn bộ website để phát hiện các liên kết bị lỗi;

- Kiểm thử khuôn dạng (form testing): Kiểm tra các kiểu dữ liệu vào, tên biến với một vài khuôn dạng định trước;

- Kiểm thử các giao dịch dạng "điểm cuối tới điểm cuối": Tương tự như kiểm tra các đường dẫn hoàn chỉnh trong kiểm thử luồng dữ liệu;

- Kiểm thử các thành phần Java: Java applets là các ứng dụng thực thi tại phía Client hay các ứng dụng java khác chạy trên Server, những kiểu ứng dụng này cũng cần phải được kiểm thử tương tự như các ứng dụng khác.

Bên cạnh những dạng kiểm thử chức năng trên, một số dạng kiểm thử liên quan tới các kỹ thuật kiểm thử cũng được sử dụng cho các ứng dụng web như:

- Kiểm thử tải: Kiểm tra để bảo đảm về thời gian đáp ứng;

- Kiểm thử khả năng sử dụng: Tập trung vào việc kiểm tra xem thiết kế của website là dễ hay khó sử dụng;

- Kiểm thử browser: Kiểm tra sự tương thích với các browsers khác nhau.

8.4. Thiết kế các ca kiểm thử

Mục tiêu thiết kế ca kiểm thử nhằm:

- Tìm ra nhiều lỗi nhất;

- Với chi phí và thời gian ít nhất.

Trong các thập kỷ 80-90 người ta đã nghiên cứu nhiều loại phương pháp thiết kế ca kiểm thử. Trong các phương pháp này, phương pháp thiết kế được chọn theo cơ chế:

- *Bảo đảm tính đầy đủ (không sót phần nào);*
- *Cung cấp khả năng phát hiện lỗi nhiều nhất.*

Việc thiết kế 1 ca kiểm thử được đặt ra với lý do sau là chủ yếu:

- Số con đường lôgic*/mạch thực hiện trong chương trình/* là rất lớn;

- Nhiều trạng thái dữ liệu khác nhau: số đại lượng, giá trị, sự thay đổi trong tiến trình, sự kết hợp giữa chúng.

Một trong những vấn đề quan trọng nhất trong kiểm thử phần mềm là thiết kế và tạo ra các *ca kiểm thử (Test case) có hiệu quả* (ca kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất).

Với những ràng buộc về thời gian và chi phí đã cho, thì vấn đề then chốt của kiểm thử là phải trả lời câu hỏi: *Tập con nào của tất cả ca kiểm thử có thể có khả năng tìm ra nhiều lỗi nhất?*

Thông thường, phương pháp kém hiệu quả nhất là kiểm tra tất cả đầu vào ngẫu nhiên - quá trình kiểm thử một chương trình bằng việc chọn ngẫu nhiên một tập con các giá trị đầu vào có thể.

Về mặt khả năng tìm ra nhiều lỗi nhất, tập hợp các ca kiểm thử được chọn ngẫu nhiên có rất ít cơ hội là tập hợp tối ưu hay gần tối ưu.

Sau đây là một số phương pháp để chọn ra một tập dữ liệu kiểm thử một cách thông minh.

Để kiểm thử hộp đen và kiểm thử hộp trắng một cách thấu đáo là không thể. Do đó, người ta chọn ra một chiến lược kiểm thử hợp lý là chiến lược có thể kết hợp sức mạnh của cả hai phương pháp trên.

Chiến lược kiểm thử hợp lý: Phát triển một cuộc kiểm thử nghiêm ngặt vừa bằng việc sử dụng các phương pháp thiết kế ca kiểm thử theo hướng hộp đen nào đó và sau đó bổ sung thêm những ca kiểm thử này bằng việc khảo sát tính logic của chương trình, sử dụng phương pháp hộp trắng.

Bảng 8.4. Các kỹ thuật phân theo chiến lược

Hộp đen	*Hộp trắng*
1. Phân lớp tương đương	1. Bao phủ câu lệnh
2. Phân tích giá trị biên	2. Bao phủ quyết định
3. Đồ thị nguyên nhân - kết quả	3. Bao phủ điều kiện
4. Đoán lỗi	4. Bao phủ điều kiện - quyết định
	5. Bao phủ đa điều kiện.

Mỗi phương pháp có những ưu điểm cũng như khuyết điểm riêng, do đó để có được tập các ca kiểm thử tối ưu, chúng ta cần kết hợp hầu hết các phương pháp.

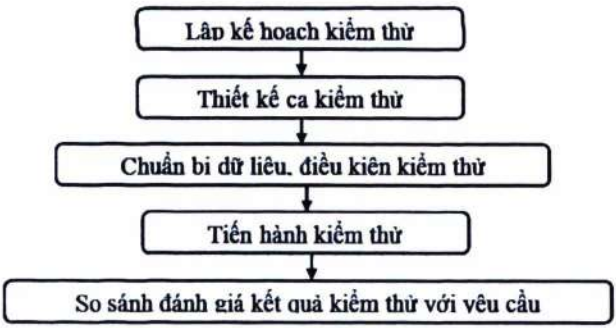
Quy trình thiết kế các ca kiểm thử sẽ bắt đầu bằng việc phát triển các ca kiểm thử sử dụng phương pháp hộp đen và sau đó phát triển bổ sung các ca kiểm thử cần thiết với phương pháp hộp trắng.

Kiểm thử hộp trắng có liên quan tới mức độ mà các ca kiểm thử thực hiện hay bao phủ tính logic (mã nguồn) của chương trình.

Kiểm thử hộp trắng cơ bản là việc thực hiện mọi đường đi trong chương trình, nhưng việc kiểm thử đầy đủ đường đi là một mục đích không thực tế cho một chương trình với các vòng lặp.

8.5. Tiến trình kiểm thử

Kiểm thử thường được thực hiện theo tiến trình sau:



Hình 8.23. Tiến trình kiểm thử

Câu hỏi đặt ra là khi nào thì kiểm thử xong? Làm thế nào để biết rằng kiểm thử đã đủ? Về nguyên tắc:

- Không bao giờ kiểm thử được tất cả;

- Vận hành chương trình là đang kiểm thử;

- Kiểm thử tiếp tục khi chương trình còn hoạt động.

Kỹ sư phần mềm cần các tiêu chuẩn nghiêm ngặt để xác định có cần phải tiếp tục kiểm thử không.

Musa và Ackerman đã đưa ra khái niệm về *kiểm thử đủ*:

- Không bao giờ có thể chắc chắn rằng một phần mềm nào đó là không có sai;

- Khi thực hiện đầy đủ các kiểm thử thì "xác suất để CPU vận hành 1000 giờ vô lỗi trong một môi trường xác định là không nhỏ hơn 0.995" với độ tin tưởng là 95%.

8.6. Các công cụ kiểm thử

Để bảo đảm chất lượng phần mềm, hoạt động kiểm thử giữa vai trò rất quan trọng song hoạt động này lại tiêu tốn và chiếm tỷ trọng khá lớn công sức và thời gian trong một dự án. Do vậy, nhu cầu tự động hóa quy trình kiểm thử phần mềm được đặt ra.

Công cụ kiểm thử (Test tool) là công cụ giúp thực hiện việc kiểm thử phần mềm một cách tự động. Tất nhiên không phải lúc nào cũng dùng được nó. Thường nó chỉ dùng trong các trường hợp sau:

1- Không đủ tài nguyên (số ca kiểm thử quá nhiều, ví dụ như kiểm thử chức năng của một website)

2- Kiểm thử hồi quy (với nhiều phiên bản liên tiếp hàng ngày, phải kiểm thử cả phần mới sửa và phần không sửa)

3- Kiểm thử khả năng vận hành phần mềm trong môi trường đặc biệt (đo tốc độ trung bình xử lý một yêu cầu của web server,...).

Việc phát triển kiểm thử tự động cũng tuân theo các bước phát triển phần mềm. Nó giống như phát triển một dự án. Quy trình kiểm thử tự động trải qua các khâu:

1- Lập kế hoạch kiểm thử;

2- Thiết kế ca kiểm thử;

3- Phát triển test script (lệnh đặc tả);

4- Thực hiện kiểm thử tự động;

5- Báo cáo kết quả;

6- Đánh giá kết quả.

Dưới đây chúng ta sẽ thống kê một số công cụ kiểm thử điển hình như sau:

1- Công cụ kiểm thử tự động Quick test professional

Trong số các công cụ kiểm thử tự động thương mại nổi tiếng như Quick test professional, Winrunner, Rational Robot, LoadRunner, Selenium, TestMaker, Rational Robot, SilkTest, Nunit, Jtest,... thì Quick test professional phiên bản 10.0 của hãng HP là khá tốt bao gồm được nhiều chức năng điển hình của một công cụ kiểm tra tự động.

Ngoài khả năng kiểm tra chức năng, Quick test professional còn cho phép thực hiện kiểm tra hồi quy một cách tự động. Đây là công cụ áp dụng phương pháp Keyword-Driven, một kỹ thuật scripting hiện đại, cho phép kiểm thử viên bổ sung test case bằng cách tạo file mô tả cho nó mà không cần phải chỉnh sửa hay bổ sung bất cứ scrip nào cả. Nó cũng phù hợp trong tình huống chuyển giao công việc mà người mới tiếp nhận chưa có thời gian hoặc không hiểu scrip vẫn có thể thực hiện kiểm tra phần mềm theo đúng yêu cầu [4].

2- Kiểm thử đơn vị với JUnit

JUnit là một framework đơn giản dùng cho việc tạo các Unit testing tự động và có thể chạy các ca kiểm thử lặp đi lặp lại. Nó chỉ là một phần của họ kiến trúc XUnit cho việc tạo ra các Unit testing. JUnit là một chuẩn trên thực tế cho unit testing trong Java.

JUnit là công cụ giúp ta thử nghiệm, gỡ rối chương trình Java. Nhờ JUnit, chúng ta dễ dàng theo dõi diễn biến của chương trình, nhanh chóng dàn dựng hàng loạt ca kiểm thử (test case) để kiểm tra mọi việc có xảy ra đúng như dự định không. Ban đầu JUnit được Kent Beck và Erich Gamma

xây dựng, sau đó giới lập trình viên java biến JUnit thành một đề án nguồn mở. Ngày nay nó trở thành một "công cụ chuẩn" mà mọi lập trình viên Java đều nên biết cách dùng.

Một số đặc điểm quan trọng của JUnit:

- Xác nhận việc kiểm tra kết quả được mong đợi;

- Các test suite cho phép chúng ta dễ dàng tổ chức và chạy các test;

- Hỗ trợ giao diện đồ họa và giao diện dòng lệnh.

Các kiểm thử trong JUnit có thể là các kiểm thử được chấp nhận hay thất bại. Chúng được thiết kế để có thể tự chạy mà không cần con người. Từ những thiết kế như vậy, chúng ta có thể thêm các bộ kiểm thử vào quá trình tích hợp và xây dựng phần mềm một cách liên tục và để cho các kiểm thử tự động chạy [4].

3- Thư viện kiểm thử NUnit

Giống như JUnit, NUnit là một thư viện lập trình được xây dựng nhằm hỗ trợ cho việc kiểm thử khi làm việc với các ngôn ngữ trên nền Dot Net của Microsoft. NUnit là một thư viện mã nguồn mở, được xây dựng dựa trên ý tưởng và phong cách của JUnit. Chính vì thế mà đa số các lớp và phương thức trong JUnit đều có thể tìm thấy trong NUnit. Tuy nhiên, do sự khác biệt về ngôn ngữ nên chúng ta cần đặc biệt chú ý khi làm việc trên môi trường Dot Net.

NUnit có hai môi trường làm việc, một là trong môi trường dòng lệnh (command line), một là trong môi trường đồ hoạ.

Do NUnit là một thư viện mở rộng nên trước khi sử dụng chương trình chúng ta phải khai báo và tham chiếu tới nó. Tuỳ vào ngôn ngữ sử dụng mà chúng ta có cấu trúc khác nhau. Trong Dot Net khi muốn sử dụng thư viện NUnit chúng ta phải xây dựng một lớp riêng, chúng ta sử dụng lớp attribute để đánh dấu lớp này và sẽ được thư viện NUnit thực thi trong quá trình kiểm thử.

Sau khi chuẩn bị tất cả các công việc cần thiết cho đoạn mã test như mong muốn, chúng ta phải biên dịch chương trình nguồn dưới dạng DLL. Sau

đó sử dụng NUnit để tiến hành công việc kiểm thử. NUnit chỉ làm việc với các file DLL, nó sẽ tiến hành phân tích cấu trúc để biết được các tài nguyên lưu trữ trong thư viện đó như các đối tượng, các phương thức, các biến, các sự kiện....

Bên cạnh những thao tác đơn giản, NUnit còn cung cấp cho chúng ta nhiều tuỳ chọn khác nhau trong quá trình viết các ca kiểm thử cho chương trình. Chúng ta chỉ thực thi việc kiểm thử đối với phương thức được thông báo là thất bại trong lần kiểm thử trước đó, đồng thời bỏ qua những ca kiểm thử đã thành công để tiết kiệm thời gian.

4- Thư viện kiểm thử CppUnit

CppUnit được viết cho ngôn ngữ C++, lấy cơ sở từ thư viện JUnit. Do mỗi ngôn ngữ có sự khác biệt về ngôn ngữ cú pháp và cách sử dụng tài nguyên trong chương trình, đặc biệt là có bộ máy biên dịch khác nhau, chính vì vậy tuy cùng xây dựng dựa trên thư viện JUnit nhưng NUnit và CppUnit có cách sử dụng hoàn toàn khác nhau. Trong C++ có file Header, chúng ta có thể sử dụng file này để định nghĩa những phương thức cần thiết cho quá trình kiểm thử. Chúng ta sẽ xây dựng một một lớp kế thừa từ lớp TestFixture của CppUnit, sau đó định nghĩa các phương thức cho lớp đó.

Để tạo một ví dụ kiểm thử đơn giản, chúng ta phải làm như sau:

- Lớp con TestCase. Viết đè phương thức *runTest()*. Khi muốn kiểm tra một giá trị, gọi hàm CPPUNIT_ASSERT(bool)

5- LoadRunner

Mercury LoadRunner là công cụ kiểm thử tự động thực hiện việc kiểm tra hiệu năng của phần mềm. Nó cho phép chúng ta tìm ra những lỗi về khả năng thực thi bằng việc phát hiện nguyên nhân, chỗ làm cho phần mềm chạy chậm hoặc không đúng yêu cầu. Đây là công cụ mạnh với giải pháp kiểm tra tải, phát hiện và đưa ra giải pháp cải tiến.

Ứng dụng LoadRunner sẽ giúp giảm thời gian viết test script đến 80%, đó là nhờ nó cung cấp chức năng tự động phát sinh script mô tả lại các tình huống muốn kiểm tra.

LoadRunner có khả năng tạo ra hàng ngàn người dùng ảo thực hiện các giao dịch cùng một lúc. Sau đó LoadRunner giám sát các thông số xử lý của

phần mềm được kiểm tra. Kết quả thống kê sẽ được lưu lại và cho phép kiểm thử viên thực hiện phân tích.

6- Selenium

Selenium gồm một bộ các công cụ hỗ trợ kiểm tra tự động tính năng của ứng dụng web, bao gồm: Selenium IDE, Selenium Remote Control (RC), Selenium Core và Selenium Grid. Bộ công cụ này hoạt động tốt trên nhiều hệ điều hành như Windows, Linux, Mac... và có thể thực thi kiểm tra trực tiếp trên các trình duyệt web khác nhau: Firefox, Internet Explorer, Safari,... Do Selenium là một công cụ hỗ trợ kiểm tra tính năng nên Selenium không có khả năng giả lập nhiều người dùng ảo cùng một lúc. Công việc của nó là chạy kiểm tra tự động dựa trên một kịch bản đã được thiết kế từ trước. Qua đó chúng ta có thể chắc chắn rằng đối tượng kiểm tra có hoạt động đúng như mong đợi hay không.

7-TestMarker

Là công cụ hỗ trợ kiểm tra hiệu năng và tính năng của ứng dụng web. TestMaker có khả năng chạy test script được thâu từ các công cụ khác như TestGen4Web và Selenium. Tuy nhiên, phiên bản được xây dựng sẵn để cài đặt và sử dụng miễn phí nhưng bị hạn chế khả năng chạy kiểm tra tối đa với 200 người dùng ảo. Để xóa bỏ hạn chế này, người sử dụng có thể trả tiền hoặc tự tải mã nguồn của TestMaker về và tự xây dựng (build) thành một bộ cài đặt và sử dụng hoàn chỉnh. Điều này đòi hỏi kiến thức về Java, Ant, NetBean, ...

Bên cạnh các thư viện đã giới thiệu hỗ trợ cho quá trình kiểm thử phần mềm ngay trong quá trình viết mã, còn có rất nhiều các thư viện khác được viết ra với mục đích tương tự nhưng áp dụng cho các ngôn ngữ khác nhau. Ngay cả các ngôn ngữ viết cho Web cũng có nhiều công cụ kiểm thử riêng của mình. Trong quá trình viết mã, cần có các plug-in được xây dựng cho các công cụ hỗ trợ. Điều này giúp cho lập trình viên rất nhiều khi xây dựng các ca kiểm thử cho các đoạn mã của họ. Như JUnit có bộ plug in được sử dụng cho NetBean, Jbuilder thì NUnit có plugin TestDriven được sử dụng trong bản Visual Studio, chúng ta có thể chạy ngay các ca kiểm thử trong môi trường soạn thảo mà không cần biên dịch chương trình.

TÓM TẮT CHƯƠNG

Chiến lược kiểm thử là sự tích hợp các kỹ thuật thiết kế ca kiểm thử tạo thành một dãy các bước nhằm hướng dẫn quá trình kiểm thử phần mềm thành công.

Mô hình chiến lược kiểm thử tổng thể được đề xuất theo hình chữ V, bao gồm rà soát mã, kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống và kiểm thử chấp nhận, tương ứng với tiến trình phát triển phần mềm.

Trong tiến trình kiểm thử, người ta chia ra thành nhiều mức kiểm thử. Về phương diện thực hành, người ta thường chia thành 4 mức cơ bản: Kiểm thử đơn vị; Kiểm thử tích hợp; Kiểm thử hệ thống và Kiểm thử chấp nhận/ thẩm định.

Có 2 tiến trình thực hiện kiểm thử:

1-Tiến trình thực hiện kiểm thử tương ứng với tiến trình phát triển (theo từng mô hình);

2-Tiến trình kiểm thử thường theo mô hình *chiến lược kiểm thử tổng thể*.

Có rất nhiều phương pháp thiết kế trường hợp kiểm thử cho phần mềm. Những phương pháp này cung cấp cho người phát triển một cách tiếp cận hệ thống tới việc kiểm thử. Quan trọng hơn, những phương pháp này đưa ra một cơ chế có thể giúp đảm bảo tính đầy đủ của kiểm thử và đưa ra khả năng cao nhất để phát hiện ra lỗi trong phần mềm.

Bất kỳ sản phẩm kỹ nghệ nào đều có thể được kiểm thử theo một trong hai cách:

1- *Kiểm thử chức năng/hộp đen: cho dữ liệu đầu vào đúng/sai, kiểm tra đầu ra đúng /sai, tức là kiểm thử xem từng chức năng có vận hành đúng không, không quan tâm đến cấu trúc bên trong của chức năng đó;*

2- *Kiểm thử cấu trúc/hộp trắng*: không những quan tâm đến mối quan hệ giữa đầu vào và đầu ra của chức năng đó mà còn quan tâm đến cấu trúc bên

trong, quan tâm chi tiết đến từng đầu vào đầu ra của các thành phần cấu thành trong đó và cả sự ăn khớp giữa chúng nữa, tức là bảo đảm rằng sự vận hành bên trong thực hiện đúng theo đặc tả và tất cả các thành phần bên trong đều được quan tâm và được kiểm tra một cách chi tiết.

Trong quá trình kiểm thử, cần tuân theo nguyên tắc:

- Không bao giờ kiểm thử được tất cả;
- Vận hành chương trình là đang kiểm thử;
- Kiểm thử tiếp tục khi chương trình còn hoạt động.

CÂU HỎI CHƯƠNG 8

8.1. Mỗi chiến lược đáp ứng được yêu cầu người quan tâm thì cần có yêu cầu gì?

8.2. Việc lựa chọn chiến lược kiểm thử phụ thuộc vào những yếu tố nào?

8.3. Thực chất của kiểm thử hộp đen và kiểm thử hộp trắng? Nêu rõ một số đặc điểm so sánh giữa chúng?

8.4. Trong chiến lược kiểm thử hệ thời gian thực, việc tích hợp phần cứng và phần mềm nhằm tìm lỗi trên các phương diện nào?

8.5. Thiết kế ca kiểm thử đơn vị bắt đầu khi nào?

8.6. Kiểm thử tích hợp từ trên xuống có thuận lợi và khó khăn gì?

8.7. Nhược điểm chính của cách tiếp cận trên xuống là gì?

8.8. Khi việc kiểm thử tích hợp được tiến hành, người kiểm thử phải xác định "mô-đun găng". Một mô-đun găng cần có những đặc trưng nào?

8.9. Khi thiết kế hệ thống, vấn đề an ninh quan tâm đến các khía cạnh nào?

8.10. Một ca kiểm thử được lựa chọn tốt nên có những tính chất nào?

BÀI TẬP CHƯƠNG 8

Bài tập 8.1

Xác định các ca kiểm thử nhờ kỹ thuật đồ thị dòng cho một đặc tả chương trình sau:

```
Begin
Write(Nhap vao tuoi cua mot nguoi:');
Readln(tuoi);
Case tuoi Of
1..17:Writeln(Nguoi nay la thieu nien');
18..39:        Writeln(Nguoi nay la thanh nien');
40..60: Writeln(Nguoi nay la trung nien');
    Else Writeln(Nguoi nay la lao nien');
End;
Readln;
End.
```

Bài tập 8.2

Xác định các ca kiểm thử nhờ kỹ thuật đồ thị dòng cho một đặc tả chương trình sau:

```
Begin
Clrscr; Write ('Nhap vao gia tri cua N :');
Readln(N);
S:=0; i:=1;
```

Repeat

S:=S+i; i:=i+1;

Until *i>N;*

Writeln('Ket qua la ',S); Readln;

End.

Bài tập 8.3

Xác định các ca kiểm thử nhờ kỹ thuật đồ thị dòng cho một đặc tả chương trình sau:

```
Readln (a,b);
IF  b = 0 THEN
    Begin
      d := a; x := 1; y := 0; writeln (d, x, y);
    End
        ELSE
     Begin
      x2 := 1; x1 := 0; y2 := 0; y1 := 1;
      While b > 0 Do
         Begin
           q := a div b; r := a mod b; x := x2 - qx1; y := y2 - qy1;
           a := b; b := r; x2 := x1; x1 := x; y2 := y1; y1 := y;
         End;
       d := a; x := x2 ; y :=y2; Writeln(d, x, y);
     end;
 ENDIF.
```

Bài tập 8.4

Xác định các ca kiểm thử nhờ kỹ thuật đồ thị dòng cho một đặc tả chương trình sau:

Do while còn bản ghi học viên chưa xử lý trong tệp CAOHOC-CNTT

Read bản ghi học viên tiếp theo

If điểm thi môn Toán >=5 AND điểm thi môn Tin >=5

Then gán tên học viên vào danh sách đỗ;

Else If tổng điểm thi của học viên với điểm ưu tiên >= 10

Then gán tên học viên vào danh sách đỗ;

Else gán tên học viên vào danh sách thi lại;

Endif

Endif

Enddo

Chương 9
NÂNG CẤP PHẦN MỀM

Việc quản lý thay đổi và nâng cấp phần mềm có tầm quan trọng rất lớn đối với một tổ chức. Hoạt động bảo trì không chỉ giới hạn các thay đổi phần mềm mà còn có cả các thay đổi phần cứng và các thay đổi thủ tục nghiệp vụ. Bảo trì là giai đoạn cuối cùng trong tiến trình kỹ nghệ phần mềm, nó tiêu tốn rất nhiều thời gian, công sức và chi phí. Tái kỹ nghệ là công nghệ đặc trưng giúp cho việc bảo trì các hệ thống phần mềm hiệu quả và nhanh chóng.

Chương này trình bày các nội dung cơ bản trong quá trình bảo trì và nâng cấp phần mềm.

Học viên cần nắm vững khái niệm về bảo trì, nâng cấp và tái kỹ nghệ hệ thống, biết thực hiện tiến trình nâng cấp phần mềm, cài đặt thay đổi, thực hiện tiến trình tái kỹ nghệ.

9.1. Bảo trì phần mềm

9.1.1. Khái niệm về bảo trì phần mềm

Bảo trì là giai đoạn cuối cùng của tiến trình phát triển phần mềm và chiếm tỷ lệ rất lớn nỗ lực chi tiêu của tổ chức phần mềm. Khi HTTT được cài đặt, thực chất hệ thống đã bắt đầu vào pha bảo trì. Tại giai đoạn này, một bộ phận lập trình viên có trách nhiệm thu nhập và xử lý các yêu cầu bảo trì từ phía người dùng.

Nguyên nhân chính dẫn đến bảo trì phần mềm là nhằm khắc phục các lỗi xử lý bên trong được phát hiện khi sử dụng phần mềm, môi trường nghiệp vụ đã thay đổi, những yêu cầu mới xuất hiện, máy tính và các thiết bị mới cần được bổ sung vào hệ thống, hiệu năng hoặc độ tin cậy của hệ thống cần được cải thiện. Các hệ thống cần được bảo trì nếu chúng ta muốn chúng là những phần hữu ích trong môi trường nghiệp vụ.

Bảo trì phần mềm đồng nghĩa với việc sửa chữa hoặc nâng cấp/hoàn thiện phần mềm để duy trì sự sống cho phần mềm, đáp ứng được những yêu cầu thay đổi của môi trường phát triển. Chúng ta hình dung khái niệm bảo trì thông qua bốn hoạt động cần thực hiện sau khi chương trình đó được đưa ra dùng [23]:

1- *Bảo trì sửa chữa*: là tiến trình bảo trì đầu tiên, bao gồm việc chẩn đoán và sửa chữa các lỗi. Hoạt động bảo trì này xuất hiện đầu tiên với giả thiết rằng việc kiểm tra phần mềm khó có thể bảo đảm việc phát hiện ra tất cả các lỗi của hệ thống. Trong khi sử dụng sẽ xuất hiện các lỗi. Bảo trì sửa chữa thường có trong giai đoạn bảo hành có thời hạn được đặt ra khi bắt đầu dự án.

2- *Bảo trì thích nghi:* là các nâng cấp sửa đổi phần mềm để đáp ứng với sự thay đổi của môi trường do sự phát triển nhanh chúng của các thiết bị, phần cứng.

3- *Bảo trì hoàn thiện:* là các hoạt động sửa đổi chức năng hiện tại và thêm các chức năng mới nâng cao do các yêu cầu nghiệp vụ mới hoặc sửa đổi chức năng đang dùng cho phù hợp với chính sách, quy trình thích hợp,... Đời sống có ích của một phần mềm thường kéo dài hơn 10 năm, vì vậy có thể nói hoạt động bảo trì hoàn thiện là chủ yếu và chiếm đại đa số công sức, chi phí dành cho bảo trì phần mềm.

4- *Bảo trì phòng ngừa:* xuất hiện khi phần mềm được thay đổi để cải thiện tính bảo trì hay tin cậy trong tương lai, hay đưa ra một cơ sở tốt hơn cho việc nâng cao trong tương lai. Hoạt động này được đặc trưng bởi việc các kỹ thuật "*kỹ nghệ ngược*" và "*tái kỹ nghệ*".

Như vậy, các hoạt động cơ bản của bảo trì là thu nhận yêu cầu, chuyển yêu cầu thành thay đổi, thiết kế thay đổi, thực hiện thay đổi.

Thay đổi là điều không thể tránh khỏi của hệ thống và thế giới thực, nhất là các hệ thống phần mềm đã xây dựng. Nó thường được cài đặt bằng cách điều chỉnh những thành phần đang tồn tại và bổ sung những thành phần mới cho hệ thống. Vì vậy cần có cơ chế để đánh giá, kiểm soát và thực hiện được những thay đổi đó.

Các hoạt động bảo trì dường như không có điểm kết thúc và nói chung chỉ kết thúc khi phần mềm không được sử dụng nữa hoặc một phần mềm mới

sẽ thay thế hệ thống đang dùng. Việc quyết định bảo trì hệ cũ hay xây dựng hệ mới phụ thuộc nhiều vào tài chính của tổ chức. Chi phí sửa lỗi phần mềm thường là thấp nhất, còn chi phí bổ sung hoặc điều chỉnh chức năng là lớn nhất và rất lớn so với các chi phí bảo trì khác.

Ngày nay, việc phát triển cũng như quá trình bảo trì các hệ thống phần mềm được hỗ trợ nhiều bởi các công cụ, đó là kỹ nghệ phần mềm có máy tính trợ giúp (Computer Aided Software Engineering- CASE). Công nghệ CASE đang phát triển mạnh mẽ, bao gồm các công cụ về: lập kế hoạch hệ thống, quản lý dự án, các công cụ trợ giúp phân tích và thiết kế, cài đặt hệ thống, tích hợp và kiểm thử, làm bản mẫu...

Các công cụ bảo trì phần mềm có thể được chia theo các chức năng sau:

- *Kỹ nghệ ngược với các công cụ đặc tả*: nhận chương trình gốc làm đầu vào và sinh ra các mô hình phân tích và thiết kế có cấu trúc đồ thị, các thông tin thiết kế khác.

- *Công cụ tái cấu trúc và phân tích mã*: phân tích cú pháp chương trình, sinh ra đồ thị luồng điều khiển, và sinh ra tự động một chương trình có cấu trúc.

- *Công cụ tái kỹ nghệ hệ thống trực tuyến*: dùng để thay đổi các hệ thống CSDL trực tuyến.

Để bảo đảm xây dựng phần mềm có thể đảm đương được với các thay đổi, người ta cần xem xét đặc trưng "*tính bảo trì được*" của phần mềm.

9.1.2. Tính bảo trì được

Tính bảo trì được có thể được định nghĩa định tính như sự dễ dùng, dễ hiểu, dễ thích nghi và dễ sửa chữa của chương trình. Các nhân tố liên quan tới môi trường phát triển về tính bảo trì là:

- Có sẵn đội ngũ phần mềm có khả năng;

- Cấu trúc hệ thống có thể hiểu được;

- Dễ xử lý hệ thống;

- Dùng ngôn ngữ lập trình đã chuẩn hóa;

- Dùng hệ điều hành đã được chuẩn hóa;

- Cấu trúc tài liệu đã được chuẩn hóa;

- Có sẵn các kịch bản kiểm tra;

- Có sẵn các trang thiết bị, máy tính phù hợp để tiến hành bảo trì.

9.1.3. Dự đoán bảo trì và dự đoán thay đổi

Dự đoán bảo trì có liên quan tới việc đánh giá những phần nào của hệ thống có thể gây ra lỗi và cần bao nhiêu chi phí để bảo trì. Khả năng chịu được sự thay đổi phụ thuộc vào khả năng bảo trì của các thành phần bị ảnh hưởng bởi sự thay đổi đó.

Chú ý rằng, thực hiện các thay đổi có thể làm hỏng hệ thống và giảm khả năng bảo trì của nó. Chi phí bảo trì phụ thuộc vào số lượng các thay đổi và chi phí thay đổi phụ thuộc vào khả năng bảo trì.

Dự đoán số lượng các thay đổi có thể xảy ra và tìm hiểu mối quan hệ giữa hệ thống và môi trường của nó. Sự thay đổi yêu cầu hệ thống có liên quan chặt chẽ tới sự thay đổi của môi trường.

9.2. Tiến trình nâng cấp phần mềm

Việc quản lý thay đổi và nâng cấp phần mềm có tầm quan trọng rất lớn đối với một tổ chức vì họ đã đầu tư một lượng vốn khá lớn vào đó. Họ rất muốn sở hữu một hệ thống hoàn hảo và bảo trì giá trị sở hữu của mình. Do vậy, tổ chức luôn chú trọng đến việc xây dựng tiến trình nâng cấp phần mềm.

Tiến trình nâng cấp phần mềm phụ thuộc vào:

- Kiểu phần mềm cần bảo trì;

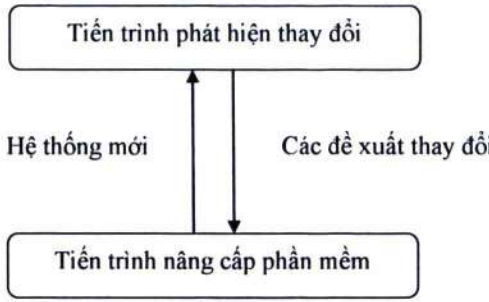
- Tiến trình phát triển phần mềm đã được sử dụng;

- Kỹ năng và kinh nghiệm của các người/tổ chức liên quan;

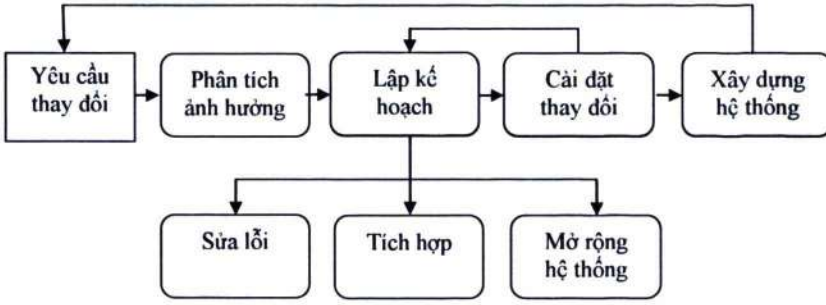
- Các đề xuất thay đổi là định hướng để nâng cấp hệ thống;

- Phát hiện thay đổi và nâng cấp được thực hiện trong vòng đời hệ thống.

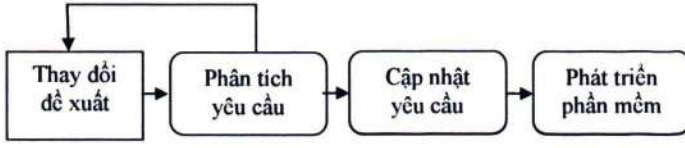
Các hình vẽ sau đây thể hiện một cách khái quát về tiến trình nâng cấp hệ thống.



Hình 9.1. Chu trình phát hiện thay đổi và nâng cấp



Hình 9.2.Tiến trình nâng cấp hệ thống



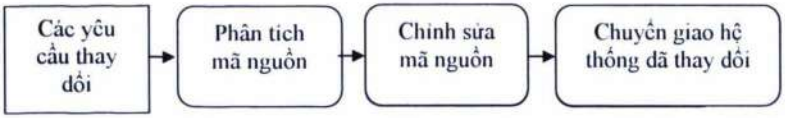
Hình 9.3. Tiến trình cài đặt thay đổi

Trên đây là những tiến trình cơ bản. Tuy nhiên, với các yêu cầu thay đổi khẩn cấp, ta có thể cài đặt chúng ngay mà không cần phải trải qua tất cả các pha của tiến trình công nghệ phần mềm. Những yêu cầu thay đổi khẩn cấp thường xảy ra khi:

- Nếu có một lỗi hệ thống nghiêm trọng xảy ra và cần phải sửa chữa;

- Nếu những thay đổi về môi trường của hệ thống gây ra những hiệu ứng không mong đợi;

- Nếu sự thay đổi về mặt nghiệp vụ yêu cầu phải có đáp ứng nhanh.



Hình 9.4. Tiến trình cài đặt thay đổi khẩn cấp

Để nâng cấp hệ thống hiện có, người ta đã đề xuất bốn chiến lược cơ bản:

1- Tách hệ thống và chỉnh sửa các quy trình nghiệp vụ;

2- Tiếp tục bảo trì hệ thống;

3- Biến đổi hệ thống bằng cách tái kỹ nghệ để nâng cấp khả năng bảo trì của nó;

4- Thay thế hệ thống bằng một hệ thống mới.

Việc lựa chọn chiến lược nâng cấp hệ thống phụ thuộc vào chất lượng hệ thống và nghiệp vụ của người dùng.

Các loại hệ thống hiện có được phân loại dựa trên tiêu chí chất lượng và giá trị nghiệp vụ mà nó mang lại như sau:

- Chất lượng thấp và giá trị nghiệp vụ thấp: những hệ thống này nên được tách ra;

- Chất lượng thấp và giá trị nghiệp vụ cao: những hệ thống này có giá trị nghiệp vụ cao nhưng chi phí bảo trì khá lớn. Do vậy, nên tái kỹ nghệ hoặc thay thế bởi một hệ thống thích hợp;

- Chất lượng cao và giá trị nghiệp vụ thấp: nên thay thế bằng các thành phần có giá trị nghiệp vụ cao;

- Chất lượng cao và giá trị nghiệp vụ cao: tiếp tục sử dụng và bảo trì hệ thống theo cách thông thường.

Việc đánh giá giá trị nghiệp vụ được thực hiện từ nhiều khung nhìn khác nhau. Nên phỏng vấn các người có liên quan khác nhau và đối sánh kết quả thu được. Các người có liên quan thường là:

- NSD cuối;

- Khách hàng của doanh nghiệp;

- Người quản lý dây chuyền sản xuất;

- Người quản lý CNTT;

- Người quản lý cao cấp.

Việc đánh giá chất lượng hệ thống thông qua hệ thống các chỉ tiêu:

1- Quy trình nghiệp vụ: quy trình nghiệp vụ đã hỗ trợ cho các mục tiêu nghiệp vụ như thế nào?;

2- Môi trường hệ thống: môi trường hệ thống có hiệu quả như thế nào và chi phí để bảo trì nó.

3- Khả năng ứng dụng: chất lượng của ứng dụng như thế nào.

Để đo hệ thống, có thể thu thập dữ liệu định lượng để tạo ra bản đánh giá về chất lượng của hệ thống như:

- Số lượng các yêu cầu thay đổi của hệ thống;

- Số lượng các giao diện người dùng khác nhau;

- Số lượng dữ liệu được sử dụng trong hệ thống.

9.3. Tái kỹ nghệ hệ thống

9.3.1. Khái niệm tái kỹ nghệ hệ thống

Bảo trì là giai đoạn cuối cùng trong tiến trình kỹ nghệ phần mềm, nó tiêu tốn rất nhiều thời gian, công sức và chi phí. Tái kỹ nghệ là công nghệ đặc trưng giúp cho việc bảo trì các hệ thống phần mềm hiệu quả và nhanh chóng.

Chúng ta sẽ phải tạo ra một sản phẩm mà các chức năng khác có thể được thêm vào, hiệu năng tốt hơn, độ tin cậy cao hơn và khả năng bảo trì được cải thiện (giảm chi phí bảo trì). Đó được gọi là "*tái kỹ nghệ*".

Tái kỹ nghệ giúp giảm rủi ro vì trong tiến trình phát triển phần mềm mới thì rủi ro có thể xảy ra là khá cao. Sự khác biệt cơ bản giữa tái kỹ nghệ và phát triển phần mềm mới là điểm bắt đầu cho sự phát triển.

Tái kỹ nghệ hệ thống (System re-engineering) được coi là kỹ thuật cấu trúc lại hoặc viết lại một phần hoặc toàn bộ hệ thống được thừa kế mà không thay đổi các chức năng của nó. Ở mức công ty, tái kỹ nghệ được các chuyên

gia thực hiện. Ở mức phần mềm thì tái kỹ nghệ phần mềm được các kỹ sư phần mềm thực thi.

Tiến trình tái kỹ nghệ hệ thống bao gồm nhiều hoạt động như phân tích yêu cầu tái kỹ nghệ, cấu trúc lại tài liệu, kỹ nghệ ngược, cấu trúc lại mã lệnh, cấu trúc lại dữ liệu, kỹ nghệ xuôi. Mục đích của các hoạt động này là tạo ra các phiên bản mới của các chương trình đang tồn tại, để nó có chất lượng cao hơn và bảo trì tốt hơn.

Chikofsky và Cross [20] quy ước gọi phát triển "kỹ nghệ xuôi" (Forward-Engineering) để phân biệt nó với tái kỹ nghệ (Re-Engineering). Hình 9.5 dưới đây giúp chúng ta hình dung rõ nét sự khác biệt đó.

Mô hình Forward-Engineering

Mô hình Re-Engineering

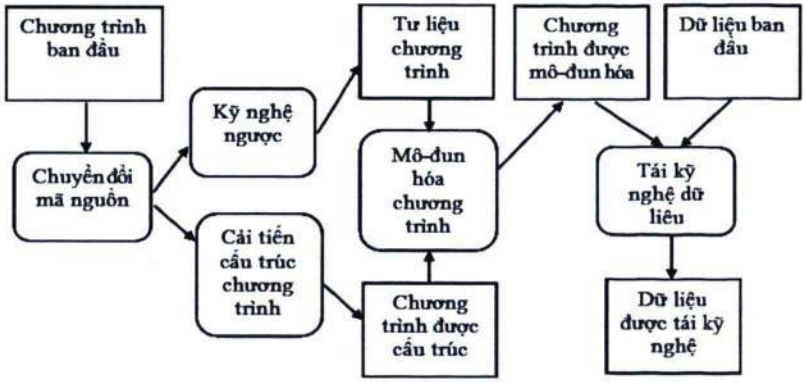
Hình 9.5. Phân biệt Forward-Engineering và Re-Engineering

Sản phẩm của tái kỹ nghệ rất đa dạng như: các mẫu phân tích, các mẫu thiết kế, các thủ tục kiểm thử,... Đầu ra cuối cùng là việc tái kỹ nghệ các tiến trình nghiệp vụ và/hoặc tái kỹ nghệ phần mềm.

Khái niệm "tái kỹ nghệ" nhen nhóm từ cuối thế kỷ 20, nhưng mối liên hệ giữa tái kỹ nghệ nghiệp vụ và tái kỹ nghệ phần mềm đã liên kết xuyên suốt trong tầm nhìn hệ thống.

Tái kỹ nghệ phần mềm thường là việc làm thực tế hoá các tiến trình kinh doanh. Hummer [20] cho rằng "Nếu các quy tắc kinh doanh thay đổi thì phần mềm cũng phải thay đổi theo".

Hình 9.6 minh họa tiến trình tái kỹ nghệ khả thi. Đầu vào của tiến trình là một chương trình kế thừa và đầu ra là một cấu trúc, phiên bản hiệu chỉnh của chương trình. Ở cùng thời điểm như chương trình tái kỹ nghệ, dữ liệu cho hệ thống có thể cũng được tái kỹ nghệ.



Hình 9.6. Minh họa về một tiến trình tái kỹ nghệ phần mềm

Trong thực tế, không ít các hệ thống phần mềm thương mại là các hệ thống cũ, nó cần để hỗ trợ cho các tiến trình nghiệp vụ. Các công ty cần các hệ thống này đến mức họ phải giữ chúng trong hoạt động. Trong chiến lược phát triển phần mềm, việc giữ lại, thay thế và phát triển kiến trúc chính là tiến trình tái kỹ nghệ phần mềm. Tái kỹ nghệ có thể còn bao gồm cả việc làm lại tài liệu hệ thống, tổ chức và cấu trúc lại hệ thống, biên dịch hệ thống sang ngôn ngữ lập trình hiện đại hơn, chỉnh sửa, cập nhật cấu trúc và dữ liệu hệ thống. Nhưng thông thường, các chức năng chính của phần mềm không thay đổi và hệ thống cấu trúc của nó cũng được giữ lại.

Từ khía cạnh kỹ thuật, tái kỹ nghệ phần mềm có thể xem như một giải pháp thứ hai cho những vấn đề của tiến hóa hệ thống. Kiến trúc phần mềm không được cập nhật như đối với các hệ thống trung tâm được phân tán. Nó cũng không thể thay đổi hoàn toàn ngôn ngữ lập trình hệ thống, vì hệ thống cũ không thể được chuyển đổi sang ngôn ngữ lập trình hướng đối tượng như Java hoặc C++ vốn có giới hạn trong hệ thống được giữ lại bởi chức năng phần mềm không thay đổi.

Tuy nhiên, từ quan điểm nghiệp vụ, tái kỹ nghệ phần mềm có thể chỉ nhằm để bảo đảm rằng hệ thống cũ có thể tiếp tục sử dụng. Nó có thể cũng đắt và gặp nhiều rủi ro như bất kỳ cách tiếp cận khác cho việc tiến hóa hệ thống.

Tái kỹ nghệ hệ thống phần mềm có ưu điểm hơn cách tiếp cận phát triển mới hệ thống, vì hai lý do:

1-Giảm rủi ro: có sự rủi ro lớn trong việc phân tích mới một hệ thống phần mềm, đó là tất yếu đối với một tổ chức. Các lỗi có thể được sinh ra trong đặc tả hệ thống, các vấn đề có thể nảy sinh, hệ thống không ổn định,...;

2-Giảm giá thành: giá thành của việc tái kỹ nghệ thấp hơn đáng kể so với giá phát triển phần mềm mới [23].

Chu kỳ tái kỹ nghệ phần mềm cũng được kết hợp với tái kỹ nghệ tiến trình nghiệp vụ. Tái kỹ nghệ tiến trình nghiệp vụ đề cập đến sự thiết kế lại các tiến trình nghiệp vụ để giảm số các hoạt động dư thừa và cải thiện các tiến trình hiệu quả. Nó thường dựa trên sự đưa vào hoặc tăng cường về cơ bản sự hỗ trợ của máy tính cho các tiến trình nghiệp vụ. Tiến trình tái kỹ nghệ thường hướng việc phát triển phần mềm như hệ thống cũ, có thể chỉ phụ thuộc vào các tiến trình đang tồn tại. Tiến trình này cần được phát hiện và hoàn thiện trước khi tiến trình tái kỹ nghệ. Vì vậy, sự cần thiết cho tái kỹ nghệ phần mềm trở lên cấp thiết trong một công ty khi sự thay đổi yêu cầu bởi tái kỹ nghệ tiến trình nghiệp vụ không thể đáp ứng được bằng việc duy trì hệ thống chương trình thông thường.

Trong quá trình tái kỹ nghệ theo cách tiếp cận từ trên xuống, sự quan tâm đầu tiên là dành cho việc tái kỹ nghệ tiến trình nghiệp vụ. Người ta đã đưa ra một số gợi ý một số nguyên tắc hướng dẫn tái kỹ nghệ tiến trình nghiệp vụ như sau [23]:

- Tổ chức theo kết quả, không theo nhiệm vụ;

- Đã có nhiều người sử dụng kết quả của tiến trình thực hiện các xử lý đó;

- Kết hợp chặt chẽ việc xử lý thông tin trong công việc thực tiễn để sản sinh các thông tin thô ban đầu;

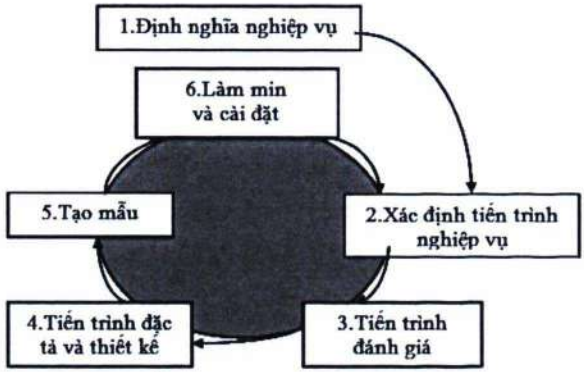
- Xử lý các nguồn lực phân tán về mặt địa lý giống như chúng đã được tập trung;

- Liên kết các hoạt động song song thay vì việc tích hợp các kết quả của chúng;

- Hướng trọng tâm vào nơi công việc được thực hiện và xây dựng điều khiển trong tiến trình đó;

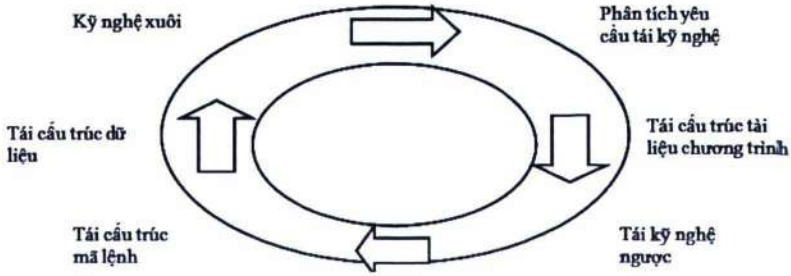
- Sưu tập dữ liệu một lần tại nguồn của chúng.

Tái kỹ nghệ tiến trình nghiệp vụ là một quá trình tiến hóa. Một *mô hình tái kỹ nghệ tiến trình nghiệp vụ* được mô tả trong hình 9.7 với 6 hoạt động.



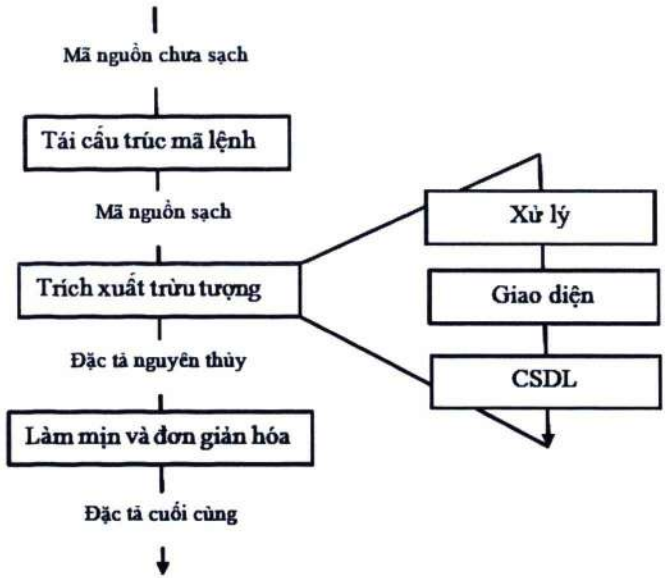
Hình 9.7. Một mô hình tái kỹ nghệ tiến trình nghiệp vụ [23]

Còn một *mô hình tiến trình tái kỹ nghệ phần mềm* cũng được mô tả với 6 hoạt động trong hình 9.8.



Hình 9.8. Một mô hình tiến trình tái kỹ nghệ phần mềm [23]

Đối với tiến trình tái kỹ nghệ ngược, tất cả các thông tin được trích rút từ mã nguồn sẽ được cung cấp cho người kỹ sư phần mềm để bảo trì. Tiến trình tái kỹ nghệ ngược được minh họa trong hình 9.9 dưới đây để biến chương trình có khả năng cấu trúc từ thiết kế thiếu cấu trúc hiện tại trở về có cấu trúc.



Hình 9.9. Tiến trình tái kỹ nghệ ngược

Cốt lõi của tiến trình kỹ nghệ ngược là hoạt động có tên gọi là "*trích xuất trừu tượng*". Người kỹ sư phải đánh giá chương trình cũ và từ mã nguồn (thường chưa là tài liệu) trích rút ra một đặc tả có ý nghĩa về tiến trình được thực hiện, về giao diện người dùng và cấu trúc dữ liệu chương trình hoặc về CSDL đã được dùng.

Giá của việc tái kỹ nghệ rõ ràng phụ thuộc vào mức độ khó khăn của công việc thực hiện. Ngoài các yếu tố liên quan đến hoạt động tái kỹ nghệ, còn có các yếu tố khác về nguyên tắc có ảnh hưởng tới chi phí tái kỹ nghệ:

1- Chất lượng phần mềm được tái kỹ nghệ: Giá trị chất lượng của phần mềm và tài liệu của nó cao hơn giá của tái kỹ nghệ;

2- *Giá trị công cụ hỗ trợ cho việc tái kỹ nghệ*: Giá trị hiệu quả cho việc tái kỹ nghệ một hệ thống phần mềm có thể thấp hơn giá trị công cụ CASE để tự động thay đổi hầu hết các chương trình;

3- *Phạm vi của yêu cầu chuyển đổi dữ liệu*: Nếu tái kỹ nghệ yêu cầu chuyển đổi một số lớn dữ liệu, điều đó làm tăng đáng kể giá thành thực hiện;

4- *Giá trị của đội ngũ chuyên môn*: Nếu trách nhiệm nhân viên bảo trì hệ thống không được sử dụng trong tiến trình tái kỹ nghệ, việc này sẽ làm tăng giá thành. Đội ngũ chuyên môn tái kỹ nghệ sẽ cần nhiều thời gian để hiểu hệ thống.

Nhược điểm chính của tái kỹ nghệ phần mềm là các giới hạn thực tế đối với việc mở rộng hệ thống có thể nằm trong phạm vi của việc tái kỹ nghệ. Ví dụ, chuyển đổi một hệ thống văn bản sử dụng cách tiếp cận chức năng sang hệ thống hướng đối tượng. Kiến trúc chính thay đổi hoặc tổ chức lại căn bản việc quản lý dữ liệu hệ thống không thể được thực hiện tự động, như thế liên quan cao đến việc thêm giá thành. Mặc dù tái kỹ nghệ có thể nâng cấp việc bảo trì, nhưng tái kỹ nghệ hệ thống sẽ không thể duy trì được như hệ thống mới phát triển bằng việc sử dụng các phương pháp tái kỹ nghệ phần mềm hiện đại.

9.3.2. Tiến trình tái kỹ nghệ phần mềm

Trên cơ sở mô hình tiến trình tái kỹ nghệ phần mềm và sau khi phân tích thấy yêu cầu tái kỹ nghệ là cần thiết, người ta đã nghiên cứu và đề xuất một tiến trình tái kỹ nghệ chi tiết bao gồm 5 hoạt động chính sau:

1- *Chuyển đổi mã nguồn* (tái cấu trúc tài liệu chương trình): chương trình được chuyển đổi từ một ngôn ngữ lập trình phiên bản cũ sang một phiên bản mới hơn hoặc sang hẳn một ngôn ngữ mới.

2- *Kỹ nghệ ngược*: phân tích chương trình để trích ra các thông tin cần thiết cho việc làm tài liệu về tổ chức và các chức năng của nó.

3- *Cải thiện cấu trúc chương trình*: Cấu trúc điều khiển của chương trình được phân tích và chỉnh sửa làm cho nó dễ đọc và dễ hiểu hơn.

4- *Mô-đun hoá chương trình*: Việc thay thế các phần của chương trình được nhóm với nhau và được làm cho phù hợp, bổ sung mô-đun mới và bỏ đi những dư thừa. Trong một số trường hợp, giai đoạn này có thể bao gồm cả sự biến đổi kiến trúc.

5- *Tái kỹ nghệ dữ liệu*: cấu trúc lại dữ liệu hệ thống tương ứng với sự thay đổi chương trình.

9.3.2.1. Chuyển đổi mã nguồn

Dạng đơn giản nhất của tái kỹ nghệ phần mềm là dịch tự động chương trình mã nguồn trong một ngôn ngữ lập trình sang mã nguồn của một số ngôn ngữ khác. Cấu trúc và tổ chức chương trình của nó không thay đổi. Ngôn ngữ đích có thể là một phiên bản mới cập nhật của ngôn ngữ gốc hoặc có thể được dịch sang một ngôn ngữ hoàn toàn khác.

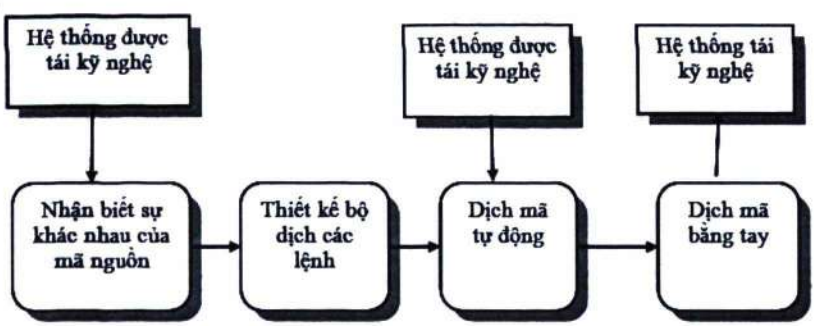
Sau đây là các lý do cần dịch mã nguồn:

1- Cập nhật nền phần cứng mới: Tổ chức muốn thay đổi nền phần cứng chuẩn. Các bộ dịch ngôn ngữ gốc không có giá trị trên phần cứng mới;

2- Thiếu đội ngũ có kỹ năng: Có thể thiếu đội ngũ bảo trì lành nghề cho ngôn ngữ gốc. Đây là một vấn đề thực tế ở đó các chương trình được viết bằng một ngôn ngữ không chuẩn mà hiện tại không sử dụng;

3- Các thay đổi chính sách của tổ chức: tổ chức có thể quyết định chuẩn hóa trên ngôn ngữ thực tế để giảm thiểu chi phí trợ giúp phần mềm trợ giúp của nó. Bảo trì một số phiên bản của bộ dịch cũ có thể rất đắt;

4- Sự yếu kém của việc trợ giúp phần mềm: Các nhà cung cấp chương trình dịch có thể đã bỏ việc kinh doanh hoặc không tiếp tục hỗ trợ sản phẩm của họ nữa.



Hình 9.10. Tiến trình dịch chương trình

Hình 9.10 chỉ ra tiến trình dịch mã nguồn. Có thể không cần hiểu chi tiết hoạt động của phần mềm hoặc chỉnh sửa kiến trúc hệ thống. Phân tích sự liên quan có thể tập trung vào ngôn ngữ lập trình tương đương như cấu trúc điều khiển chương trình.

Việc dịch mã nguồn chỉ thực sự là kinh tế nếu bộ dịch tự động sẵn sàng để thực hiện với một số lượng lớn. Đây có thể là một chương trình được viết đặc biệt, một công cụ được mua để chuyển đổi từ một ngôn ngữ này sang một ngôn ngữ khác hoặc một hệ thống mẫu thích hợp. Trong trường hợp thứ hai, cách thức một tập các lệnh để tạo sự dịch chuyển từ sự trình bày này sang một sự trình bày khác cần được viết các mẫu được tham số hóa trong ngôn ngữ nguồn được xác định và kết hợp với các mẫu tương đương trong ngôn ngữ đích.

Trong nhiều trường hợp, việc dịch hoàn toàn tự động là không thể. Các cấu trúc trong ngôn ngữ nguồn không có cấu trúc tương đương trực tiếp trong ngôn ngữ đích. Có thể các lệnh điều kiện biên dịch trong mã nguồn mà không được hỗ trợ trong ngôn ngữ đích. Trong trường hợp này, cần tạo ra sự thay đổi bằng tay làm cho phù hợp và nâng cấp hệ thống sinh lệnh.

9.3.2.2. Kỹ nghệ ngược

Kỹ nghệ ngược (Reserve-Engineering) là tiến trình phân tích phần mềm mã nguồn với mục đích chuyển đổi nó về dạng thiết kế và đặc tả. Chương trình tự nó không bị thay đổi bởi tiến trình kỹ nghệ ngược. Mã nguồn phần mềm thường có giá trị như đầu vào cho tiến trình Kỹ nghệ ngược. Tuy nhiên, điều này có thể không có và Kỹ nghệ ngược cần bắt đầu với mã thực thi.

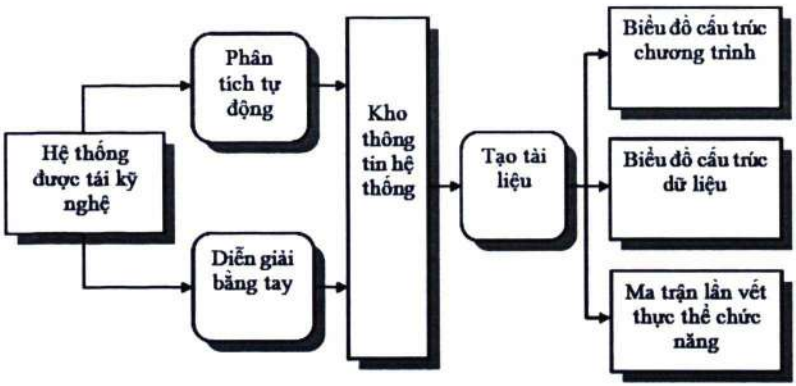
Kỹ nghệ ngược không hoàn toàn giống như tái kỹ nghệ (Re-Engineering). Mục tiêu của kỹ nghệ ngược thu được là thiết kế hoặc đặc tả của hệ thống từ mã nguồn của nó. Mục tiêu của tái kỹ nghệ là làm ra một hệ thống phần mềm mới, dễ bảo trì hơn. Tất nhiên, như chúng ta có thể thấy rằng, nhờ kỹ nghệ ngược để hiểu biết rõ hơn về hệ thống và kỹ nghệ ngược thường là một phần của tiến trình tái kỹ nghệ.

Kỹ nghệ ngược được dùng trong tiến trình tái kỹ nghệ phần mềm là để phục hồi lại thiết kế chương trình mà sẽ giúp các kỹ sư hiểu tốt một chương

trình trước khi tổ chức lại cấu trúc của nó. Tuy nhiên, không nhất thiết theo sau kỹ nghệ ngược phải là tái kỹ nghệ:

1- Bản thiết kế và đặc tả của một hệ thống đang tồn tại có thể được kỹ nghệ ngược, vì chúng có thể dùng như một đầu vào cho đặc tả các yêu cầu của chương trình cần thay thế;

2- Như một sự lựa chọn, bản thiết kế và đặc tả có thể được kỹ nghệ ngược, vì chúng sẵn sàng để giúp bảo trì chương trình. Thông tin này có thể không cần thiết cho việc tái kỹ nghệ mã nguồn hệ thống.



Hình 9.11. Tiến trình kỹ nghệ ngược

Sau khi tài liệu thiết kế hệ thống được tạo sinh, nhiều thông tin có thể được thêm vào để giúp tái tạo hệ thống đặc tả. Việc này thường liên quan nhiều tới sự diễn giải bằng tay của cấu trúc hệ thống. Đặc tả không thể được suy ra tự động từ mô hình hệ thống.

9.3.2.3. Cải thiện cấu trúc chương trình

Một số hệ thống cũ không được cấu trúc tốt do cấu trúc điều khiển bị lộn xộn với một số nhánh vô điều kiện và logic điều khiển không rõ ràng. Cấu trúc này cũng có thể bị giảm giá trị bởi sự bảo dưỡng thường xuyên. Sự thay đổi chương trình là cần thiết.

Cũng như điều khiển không có cấu trúc, các điều kiện phức tạp có thể cũng được đơn giản hóa khi một phần của chương trình được xử lý lại.

Năm 1966, Bohm và Jacopini [20] đã chứng minh rằng: bất kỳ một chương trình nào đều có thể viết lại với ngôn ngữ đơn giản là câu lệnh điều kiện *if-then-else* và vòng lặp *while* mà ở đó không dùng "câu lệnh *goto*". Định lý này là điều kiện cơ bản cho việc kiến trúc lại chương trình tự động. Con đường kiến trúc lại tự động của một chương trình gồm 2 bước: Thứ nhất - chuyển chương trình thành đồ thị có hướng, thứ hai - sinh ra một chương trình cấu trúc tương đương.

Đồ thị có hướng được sinh ra như một biểu đồ luồng chương trình, nó chỉ ra cách điều khiển di chuyển xuyên suốt chương trình. Công nghệ đơn giản hóa và biến đổi có thể được áp dụng cho đồ thị này mà không thay đổi ngữ nghĩa của nó. Công nghệ này tự dò tìm và bỏ đi những phần mã không thể tiếp cận được. Lần thứ nhất, sự đơn giản hóa được thực hiện thì một chương trình mới được sinh ra. Vòng lặp *While* và các điều kiện đơn giản được thay thế cho điều khiển *goto*. Chương trình này có thể được để trong ngôn ngữ gốc hoặc trong ngôn ngữ khác.

Vấn đề cấu trúc lại chương trình tự động bao gồm:

1- Mất các lời chú thích: Nếu chương trình có chú thích trong một dòng thì dòng chú thích này sẽ mất giá trị khi một phần của tiến trình kiến trúc lại.

2- Mất tài liệu: Tương tự, sự tương ứng giữa tài liệu chương trình bên ngoài và chương trình cũng mất. Tuy nhiên trong một vài trường hợp, cả phần chú thích và tài liệu của chương trình đều quá hạn, như vậy đây không phải là một nhân tố quan trọng.

3- Yêu cầu tính toán lớn: giải thuật nhúng trong công cụ cấu trúc lại phức tạp. Mặc dù phần cứng hiện đại, nhanh, nó cũng có thể mất một thời gian dài để hoàn thành tiến trình cấu trúc lại cho chương trình lớn.

Trong một số trường hợp, nó không thể có giá trị hiệu quả khi cấu trúc lại toàn bộ chương trình trong hệ thống. Một vài trường hợp có thể đạt chất lượng tốt hơn chương trình khác và một vài trường hợp không dễ bị thay đổi thường xuyên. Arthur [20] giả thuyết rằng, dữ liệu nên được tập hợp để giúp nhận biết

các chương trình mà nó có thể được lợi nhất từ việc cấu trúc lại. Ví dụ, các độ đo sau có thể được sử dụng để nhận biết các ứng cử cho kiến trúc lại:

- Tỉ lệ lỗi;

- Phần trăm của mã nguồn thay đổi hàng năm;

- Thành phần phức tạp.

Các nhân tố khác như mức độ mà các chương trình đó hoặc các thành phần đáp ứng được chuẩn hiện tại có thể cũng được đưa vào xem xét để quyết định cấu trúc lại.

9.3.2.4. Mô-đun hoá chương trình

Mô-đun hóa chương trình là tiến trình tổ chức lại chương trình sao cho những phần chương trình được tập hợp với nhau và được xem như là một mô-đun. Khi chương trình đã được mô-đun hóa, dễ bỏ đi những phần thừa trong các phần được thay thế, để tối ưu.

Sự tương tác của chúng và để đơn giản trong giao diện của chúng với những phần còn lại của chương trình. Ví dụ, trong một chương trình nó xử lý dữ liệu địa trấn, tất cả các hoạt động kết hợp với sự trình diễn đồ họa của dữ liệu có thể được tập hợp với nhau trong một mô-đun đơn giản. Nếu hệ thống bị phân tán, các mô-đun được tạo có thể được gói gọn như các đối tượng và truy cập thông qua giao diện chung. Vài kiểu mô-đun khác nhau có thể được tạo trong khi xử lý mô-đun hóa chương trình. Tiến trình nay gồm:

1-Trừu tượng dữ liệu: Kiểu dữ liệu trừu tượng được tạo bởi sự kết hợp dữ liệu với các thành phần tiến trình;

2- Mô-đun hóa phần cứng: Thay dữ liệu trừu tượng và tập hợp tất cả các hàm chúng với nhau một cách chặt chẽ, nó được sử dụng để điều khiển thiết bị phần cứng riêng biệt;

3-Mô-đun hóa chức năng: Đó là các mô-đun mà nó tập hợp các chức năng với nhau, nó đồng dạng hoặc các tác vụ gần nhau. Ví dụ, tất cả các chức năng có liên quan với đầu vào và giá trị đầu vào có thể được hợp nhất trong

một mô-đun đơn giản. Kiểu này của sự mô-đun hóa được xét đến khi không cần sửa lại trừu tượng dữ liệu chương trình;

4- Mô-đun trợ giúp tiến trình: Đó là các mô-đun mà ở đó tất cả các chức năng và các mục dữ liệu đặc biệt yêu cầu để trợ giúp tiến trình nghiệp vụ đặc biệt được nhóm lại. Ví dụ, trong một hệ thống thư viện, một mô-đun trợ giúp tiến trình có thể gồm tất cả các chức năng yêu cầu để trợ giúp sự phát hành và phản hồi của sách.

Mô-đun hóa chương trình thường thực hiện bằng tay bởi sự kiểm tra và sửa chữa mã nguồn. Để mô-đun hóa một chương trình, cần nhận ra quan hệ, giữa các thành phần và thực hiện những gì mà các thành phần này làm. Các công cụ trình diễn và làm trực quan trợ giúp nhưng nó không thể tự động hoàn thành tiến trình này.

Để tiết kiệm không gian nhớ, các hệ thống cũ dựa trên việc sử dụng các bảng chia sẻ và các vùng dữ liệu chung. Thông tin trong vùng này có thể truy cập và cũng có thể được sử dụng ở những phần khác nhau trong hệ thống bằng những cách thức khác nhau. Việc thay đổi những vùng dữ liệu tổng thể là chi phí rất đắt vì giá của việc phân tích dẫn đến mọi việc sử dụng dữ liệu đều thay đổi. Để giảm bớt giá thành của việc thay đổi trên các vùng dữ liệu chia sẻ này, các mô-đun chương trình xử lý có thể tập trung vào các vùng xác định của dữ liệu trừu tượng. Dữ liệu trừu tượng hoặc các kiểu dữ liệu trừu tượng tập hợp các dữ liệu lại với nhau và xử lý liên kết dẫn đến việc thay đổi ngầm. Dữ liệu trừu tượng ẩn trong các thể hiện dữ liệu và cung cấp hàm thiết lập và truy cập tới các hàm để sửa đổi và kiểm tra các dữ liệu.

Vì các giao diện không thay đổi nên những sự thay trong các kiểu dữ liệu không nên làm ảnh hưởng đến các phần khác của chương trình. Các bước có liên quan tới việc chuyển đổi từ các vùng dữ liệu chung tới các đối tượng hoặc tới các kiểu dữ liệu trừu tượng:

1- Phân tích các vùng dữ liệu chung để xác định được các trừu tượng dữ liệu logic. Nó thường xảy ra trong các trường hợp, một số trừu tượng

được kết hợp trong một vùng dữ liệu chia sẻ đơn. Nó nên được xác định và cấu trúc lại logic;

2- Tạo một kiểu hoặc đối tượng dữ liệu trừu tượng cho mỗi trừu tượng này. Nếu ngôn ngữ lập trình không có tiện ích dữ liệu ẩn, giả lập một kiểu dữ liệu trừu tượng bởi các hàm điều kiện để cập nhật và truy cập tất cả các trường của dữ liệu;

3- Sử dụng hệ thống trình duyệt chương trình hoặc bộ tham chiếu chéo để tìm tới tất cả những phần liên quan tới dữ liệu này. Thay thế cái này bằng việc gọi tới các hàm xấp xỉ.

Tiến trình này có vẻ như tốn nhiều thời gian nhưng khá dễ hiểu. Tuy nhiên, trong thực tế, nó có thể rất khác bởi cách thức trong vùng dữ liệu chia sẻ được dùng Trong phiên bản cũ của ngôn ngữ có thể có giới hạn những phương thức cấu trúc dữ liệu. Những người lập trình có thể thiết kế chiến lược quản lý dữ liệu phức tạp mà họ cần tiến hành sử dụng các mảng chia sẻ. Bởi vậy, trên thực tế mảng có thể được dùng như kiểu khác của cấu trúc dữ liệu. Vấn đề tiếp theo có nguyên nhân bởi địa chỉ gián tiếp của cấu trúc chia sẻ và địa chỉ đoạn từ một vài cấu trúc khác.

Nếu máy đích cho chương trình gốc có bộ nhớ giới hạn hay có nguyên nhân này nguyên nhân khác. Những người lập trình có thể sử dụng sự hiểu biết về thời gian sống dữ liệu và việc nhúng nó trong chương trình. Để tránh việc cấp thêm không gian, họ sử dụng miền dữ liệu tương tự để lưu các trừu tượng khác ở các điểm khác trong chương trình. Việc này có thể chỉ được khám phá sau phân tích tĩnh và phân tích động của chương trình.

9.3.25. Tái kỹ nghệ dữ liệu

Nội dung về phát triển phần mềm đều tập trung vào vấn đề thay đổi chương trình và hệ thống phần mềm. Tuy nhiên, trong một số trường hợp, lại có vấn đề tết hợp phát triển dữ liệu. Sự thay đổi cách lưu trữ, tổ chức và định dạng dữ liệu xử lý sẽ thể hiện sự thay đổi trong phần mềm mới. Tiến trình phân

tích và tổ chức lại một vài lần cấu trúc dữ liệu cũng như giá trị dữ liệu trong hệ thống làm cho hệ thống dễ hiểu hơn được gọi là *tái kỹ nghệ dữ liệu*.

Tái kỹ nghệ dữ liệu là không cần thiết nếu chức năng của hệ thống không đổi. Tuy nhiên, trong thực tế, có một số lý do dẫn đến việc sửa dữ liệu trong các chương trình của hệ thống cũ:

1- Sự suy thoái dữ liệu: Do chất lượng của dữ liệu hướng tới sự suy giảm dần. Sự thay đổi dữ liệu mở đầu cho các lỗi, các giá trị giống nhau có thể được tạo và sự thay đổi môi trường ngoài không thể được phản ánh vào trong dữ liệu. Đây là điều không thể tránh được bởi thời gian sống của dữ liệu thường là rất dài. Ví dụ, dữ liệu về cá nhân trong dữ liệu đang tồn tại khi một tài khoản được mở và có thể cần tiếp tục tồn tại ít nhất là bằng thời gian sống của khách hàng. Khi lý lịch của khách hàng thay đổi, sự thay đổi này có thể không thích hợp với dữ liệu trong nhà băng. Tái kỹ nghệ chương trình có thể làm tăng chất lượng dữ liệu và đó là điểm mấu chốt cần tới tái kỹ nghệ dữ liệu;

2- Các hạn chế vốn có được xây dựng trong chương trình: Khi thiết kế ban đầu, những người phát triển của nhiều chương trình đã đưa vào những ràng buộc tự xây dựng về số lượng dữ liệu mà nó có thể xử lý. Tuy nhiên, ngày nay chương trình thường yêu cầu xử lý nhiều dữ liệu hơn dự tính ban đầu của các nhà phát triển. Tái kỹ nghệ dữ liệu có thể được yêu cầu để hủy bỏ các giới hạn đó;

3- Phát triển kiến trúc: Nếu một hệ thống tập trung chuyển sang một kiến trúc phân tán, điều cần thiết cốt lõi của kiến trúc đó là một hệ thống quản lý dữ liệu, nó có thể được truy cập từ các máy trạm. Điều này có thể yêu cầu một sự cố gắng lớn trong việc tái kỹ nghệ để di chuyển dữ liệu từ các tệp rời vào trong hệ thống quản lý dữ liệu chủ. Sự di chuyển tới một kiến trúc chương trình phân tán có thể được khởi tạo khi một tổ chức quyết định di chuyển từ việc quản lý dữ liệu bằng các tệp cơ sở sang hệ thống quản lý CSDL.

Giống như tái kỹ nghệ chương trình, có một tập các cách tiếp cận tới tái kỹ nghệ dữ liệu. Nó phản ánh lý do tại sao việc tái kỹ nghệ dữ liệu được yêu cầu.

Bảng 9.1. Diễn tả tư tưởng tiếp cận tới việc tái kỹ nghệ dữ liệu

Tiếp cận	Diễn tả
Làm sạch dữ liệu	Các bản ghi dữ liệu và các giá trị được phân tích để nâng cấp chất lượng của chúng. Sự trùng lặp được bỏ đi, sự dư thừa thông tin được xóa bỏ và định dạng thích hợp được áp dụng cho tất cả các bản ghi. Thường những điều này không yêu cầu thay đổi chương trình.
Mở rộng dữ liệu	Trong trường hợp này, dữ liệu và chương trình được tái kỹ nghệ để bỏ đi giới hạn trên việc xử lý dữ liệu. Việc này có thể yêu cầu thay đổi tới chương trình để tăng chiều dài trường, sửa tăng giới hạn trên bảng,... Sau đó, dữ liệu, tự nó có thể viết lại và làm sạch để phản ánh sự thay đổi của chương trình.
Di chuyển dữ liệu	Trong trường hợp này, dữ liệu được di chuyển vào sự điều khiển của hệ quản trị CSDL hiện đại. Dữ liệu có thể được lưu trữ trong các tệp độc lập hoặc có thể được quản lý bởi một kiểu cũ hơn của DBMS.

Rickets [20] mô tả một số vấn đề nảy sinh với dữ liệu:

- *Vấn đề tên dữ liệu*: Tên trong chương trình cũ có thể khó hiểu vì không phân biệt rõ hiện tượng "đồng danh" hoặc "đồng nghĩa".

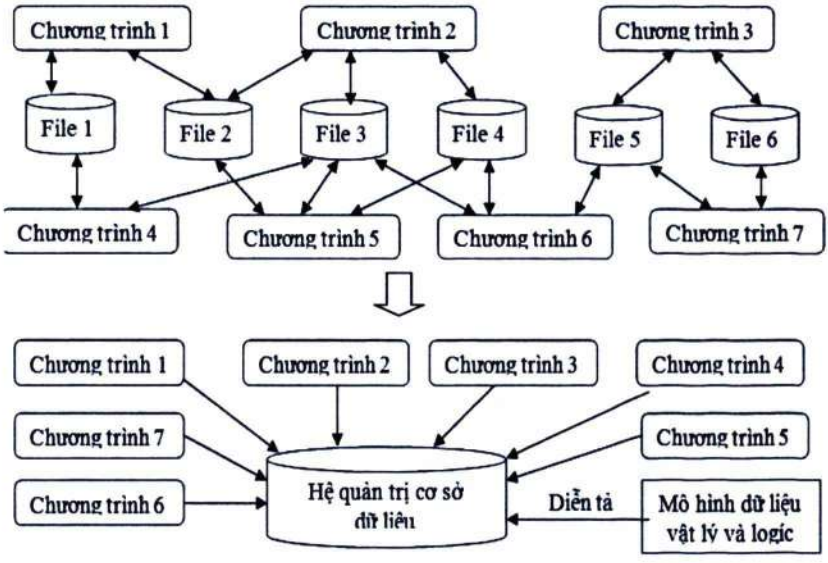
- *Vấn đề chiều dài trường*: Chiều dài trường được thiết kế không đủ cho hiện tại hoặc quá phóng đại gây đến việc lãng phí bộ nhớ.

- *Vấn đề tổ chức bản ghi*: Sự diễn tả các bản ghi tương tự thực thể có thể được tổ chức khác nhau trong các chương trình khác nhau. Đây là một vấn đề trong một số ngôn ngữ, ở đó tổ chức vật lý các bản ghi được người lập trình tạo lập và phản chiếu trong các tệp. Ngược lại, trong ngôn ngữ như C++ hoặc Java thì ở đó tổ chức vật lý của một bản ghi là trách nhiệm của trình biên dịch.

- *Các từ ngữ mã cứng*: Các giá trị từ ngữ (tuyệt đối), như tỉ lệ đòi hỏi, bao gồm trực tiếp trong chương trình tốt hơn là tham chiếu sử dụng một số tên ký hiệu.

- *Từ điển dữ liệu*: Có thể có từ điển dữ liệu định nghĩa tên sử dụng chưa nhất quán.

Định nghĩa dữ liệu và giá trị dữ liệu cũng có thể mâu thuẫn. Vì vậy, sau khi định nghĩa dữ liệu tái kỹ nghệ, giá trị dữ liệu cũng cần được chuyển đổi để phù hợp với cấu trúc mới.



Hình 9.12. Di chuyển dữ liệu

Phân tích chi tiết chương trình về sử dụng dữ liệu là việc làm cần thiết trước khi tái kỹ nghệ dữ liệu. Các phân tích này nên tập trung vào việc tìm hiểu các chức năng trong chương trình, tìm các giá trị chữ mà nó nên được thay thế với tên không đổi. Các công cụ tham chiếu qua việc phân tích các mẫu có thể sử dụng trợ giúp với các phân tích này. Một tập các bảng nên được tạo ra mà ở đó nó chỉ ra các đối tượng dữ liệu được tham chiếu và thay đổi được tạo ra với mỗi tham chiếu này.

Hình 9.13 minh họa một tiến trình tái kỹ nghệ dữ liệu. Các định nghĩa dữ liệu cần được chỉnh sửa, tổ chức lại định dạng dữ liệu và chuyển đổi các giá trị dữ liệu.

Hình 9.13. Tiến trình tái kỹ nghệ dữ liệu

TÓM TẮT CHƯƠNG

Bảo trì là giai đoạn cuối cùng của tiến trình phát triển phần mềm và chiếm tỷ lệ rất lớn nỗ lực chi tiêu của tổ chức phần mềm. Các hoạt động cơ bản của bảo trì là thu nhận yêu cầu, chuyển yêu cầu thành thay đổi, thiết kế thay đổi, thực hiện thay đổi. Các hoạt động bảo trì dường như không có điểm kết thúc.

Dự đoán bảo trì có liên quan tới việc đánh giá những phần nào của hệ thống có thể gây ra lỗi và cần bao nhiêu chi phí để bảo trì. Khả năng chịu được sự thay đổi phụ thuộc vào khả năng bảo trì của các thành phần bị ảnh hưởng bởi sự thay đổi đó.

Dự đoán số lượng các thay đổi có thể xảy ra và tìm hiểu mối quan hệ giữa hệ thống và môi trường của nó. Sự thay đổi yêu cầu hệ thống có liên quan chặt chẽ tới sự thay đổi của môi trường.

Để nâng cấp hệ thống hiện có, người ta đã đề xuất bốn chiến lược cơ bản:

1- Tách hệ thống và chỉnh sửa các quy trình nghiệp vụ;

2- Tiếp tục bảo trì hệ thống;

3- Biến đổi hệ thống bằng cách tái kỹ nghệ để nâng cấp khả năng bảo trì của nó;

4- Thay thế hệ thống bằng một hệ thống mới. Việc lựa chọn chiến lược nâng cấp hệ thống phụ thuộc vào chất lượng hệ thống và nghiệp vụ của người dùng.

Tái kỹ nghệ là công nghệ đặc trưng giúp cho việc bảo trì các hệ thống phần mềm hiệu quả và nhanh chóng. Đối tượng của hệ thống tái kỹ nghệ là nâng cấp cấu trúc hệ thống và làm cho nó dễ hiểu hơn.

Tái kỹ nghệ giúp giảm rủi ro vì trong tiến trình phát triển phần mềm mới rủi ro có thể xảy ra là khá cao. Hơn thế nữa, tái kỹ nghệ còn giúp giảm chi phí bảo trì.

Trong tiến trình tái kỹ nghệ theo cách tiếp cận từ trên xuống, sự quan tâm đầu tiên là dành cho việc tái kỹ nghệ tiến trình nghiệp vụ.

Tái kỹ nghệ tiến trình nghiệp vụ là một quá trình tiến hóa, không có điểm đầu và điểm cuối. Đối với tiến trình tái kỹ nghệ ngược, tất cả các thông tin được trích rút từ mã nguồn sẽ được cung cấp cho người kỹ sư phần mềm để bảo trì.

Tiến trình tái kỹ nghệ bao gồm các hoạt động như *Chuyển đổi mã nguồn, Kỹ nghệ ngược, Cải thiện cấu trúc chương trình, Mô-đun hoá chương trình, Tái kỹ nghệ dữ liệu.* Mục đích của các hoạt động này là tạo ra các phiên bản mới của các chương trình đang tồn tại, để nó có chất lượng cao hơn và bảo trì tốt hơn.

Kỹ nghệ ngược là tiến trình phân tích phần mềm mã nguồn với mục đích chuyển đổi nó về dạng thiết kế và đặc tả.

Kỹ nghệ ngược được dùng trong tiến trình tái kỹ nghệ phần mềm là để phục hồi lại thiết kế chương trình mà sẽ giúp các kỹ sư hiểu tốt một chương trình trước khi tổ chức lại cấu trúc của nó. Tuy nhiên, không nhất thiết theo sau kỹ nghệ ngược phải là tái kỹ nghệ.

Định nghĩa dữ liệu và giá trị dữ liệu cũng có thể mâu thuẫn. Vì vậy, sau khi định nghĩa dữ liệu tái kỹ nghệ, giá trị dữ liệu cũng cần được chuyển đổi để phù hợp với cấu trúc mới.

Phân tích chi tiết chương trình về sử dụng dữ liệu là việc làm cần thiết trước khi tái kỹ nghệ dữ liệu.

CÂU HỎI CHƯƠNG 9

9.1. Nguyên nhân chính dẫn đến bảo trì phần mềm?

9.2. Các hoạt động nào sau đây không phải là hoạt động cơ bản của bảo trì?

- *Bảo trì sửa chữa*
- *Bảo trì thích nghi*
- *Bảo trì hoàn thiện*
- *Bảo trì phòng ngừa*

9.3. Các công cụ bảo trì phần mềm có thể được chia theo các chức năng nào?

9.4. Những yêu cầu thay đổi khẩn cấp thường xảy ra khi nào?

9.5. Để nâng cấp hệ thống hiện có, người ta đã đề xuất các chiến lược cơ bản nào? Anh/chị có thể đưa ra một chiến lược nâng cấp hệ thống của riêng mình?

9.6. Tái kỹ nghệ là công nghệ đặc trưng giúp ích gì?

9.7. Vì sao nói: "Tái kỹ nghệ hệ thống phần mềm có ưu điểm hơn cách tiếp cận phát triển mới hệ thống"?

9.8. Cốt lõi của tiến trình kỹ nghệ ngược là hoạt động có tên gọi là gì?

9.9. Tiến trình tái kỹ nghệ còn thiếu hoạt động nào nữa ngoài các hoạt động sau đây:

1- *Chuyển đổi mã nguồn;*

2- *Kỹ nghệ ngược;*

3- *Cải thiện cấu trúc chương trình;*

4- *Mô-đun hoá chương trình;*

9.10. Kỹ nghệ ngược khác gì so với tái kỹ nghệ?

Chương 10
KỸ NGHỆ PHẦN MỀM HƯỚNG DỊCH VỤ

Kiến trúc hướng dịch vụ (SOA) được xem như là cuộc cách mạng mới đối với các phần mềm tích hợp ứng dụng, là một cách tiếp cận để xây dựng các ứng dụng tích hợp từ các dịch vụ độc lập với nền, các dịch vụ được tái sử dụng.

Chương này trình bày các nội dung chi tiết còn lại về kỹ nghệ phần mềm hướng dịch vụ từ phần khái quát kỹ nghệ hướng dịch vụ đã được giới thiệu trong chương 1, phần đặc tả kiến trúc đã được trình bày trong chương 6.

Học viên cần nghiên cứu kỹ các mô hình triển khai SOA, các nguyên tắc phát triển, vòng đời phát triển, các pha cơ bản, cách tạo các mẫu, các mức thực hiện kiến trúc phần mềm theo hướng này.

10.1. Kiến trúc hướng dịch vụ

10.1.1. Hướng tiếp cận dịch vụ

Thuật ngữ kiến trúc hướng dịch vụ (SOA - Service Oriented Architectural) tuy mới được phát triển từ đầu năm 2000 nhưng đang rất được quan tâm, đặc biệt khi Microsoft cho ra đời công nghệ ".NET", mà ở đó việc phát triển các dịch vụ trở nên dễ dàng hơn bao giờ hết.

Tùy theo quan điểm người dùng, họ đưa ra nhiều khái niệm khác nhau về kiến trúc hướng dịch vụ. Chẳng hạn như: "SOA là một cách tiếp cận để tổ chức các tài nguyên CNTT mà ở đó dữ liệu, logic và nguồn lực hạ tầng được truy cập thông qua các giao diện (interface) và thông điệp (Message)" [28], hoặc "SOA là một hướng tiếp cận với việc thiết kế và tích hợp các phần mềm, chức năng, hệ thống theo dạng mô-đun, trong đó mỗi mô-đun đóng vai trò là một dịch vụ có tính "kết nối lỏng", và có khả năng truy cập thông qua môi trường mạng. Hiểu một cách đơn giản thì một hệ thống SOA là một tập hợp các dịch

vụ được chuẩn hoá trên mạng trao đổi với nhau trong ngữ cảnh một tiến trình nghiệp vụ" [22].

Theo Norbert Bieberstein [22]: "*SOA là một khung công việc để tích hợp các tiến trình nghiệp vụ và hỗ trợ cơ sở hạ tầng CNTT như an ninh, các thành phần đã được chuẩn hoá- các dịch vụ. Các dịch vụ này có thể được dùng lại và có thể được phối hợp với nhau để ghi nhận các ưu tiên nghiệp vụ dễ biến động*".

Theo DotNetGuru [22], "*SOA là khái niệm về hệ thống trong đó mỗi ứng dụng được xem như một nguồn cung cấp dịch vụ*".

Xét về khía cạnh sử dụng thì "*dịch vụ*"có thể hiểu đơn giản như sau: *dịch vụ là một tập các chương trình con (thư viện) cho phép các chương trình chạy trên các máy tính khác trong mạng có thể gọi và sử dụng....*

SOA cũng có thể được hiểu là một hướng tiếp cận để xây dựng các hệ thống phân tán cung cấp các chức năng ứng dụng dưới các dạng dịch vụ tới các ứng dụng của NSD hoặc các dịch vụ khác. SOA là một kiến trúc dùng trong các chuẩn mở để biểu diễn các thành phần mềm như là các dịch vụ. Nó cung cấp một cách thức chuẩn hoá cho việc biểu diễn và tương tác với các thành phần phần mềm. Còn các thành phần phần mềm riêng lẻ trở thành các khối cơ bản để có thể sử dụng lại để xây dựng các ứng dụng khác. Các thành phần đó được sử dụng để tích hợp các ứng dụng bên trong và bên ngoài tổ chức.

Như vậy, SOA là tập hợp các dịch vụ *kết nối "mềm dẻo"* với nhau nhờ sự chuẩn hóa giao tiếp (nghĩa là một ứng dụng có khả năng giao tiếp với một ứng dụng khác mà không cần biết các chi tiết hệ thống bên trong), có giao diện được định nghĩa rõ ràng và độc lập với nền tảng của hệ thống, và có thể *tái sử dụng*. SOA là cấp độ cao hơn của sự phát triển ứng dụng, chú trọng đến quy trình nghiệp vụ và dùng giao diện chuẩn để che dấu sự phức tạp kỹ thuật bên dưới.

Mô hình Kiến trúc môi giới (Common Object Request Broker Architecture - CORBA) và mô hình Kiến trúc đối tượng thành phần phân tán (Distributed Component Object Model - DCOM) của Microsoft hay như Enterprise Java Bean (EJB) của Java cũng có kiến trúc tương tự như SOA. Tuy

nhiên, các kiến trúc cũ ràng buộc các thành phần với nhau quá chặt chẳng hạn như các ứng dụng phân tán muốn làm việc với nhau phải được thoả thuận về chi tiết tập hàm API, một thay đổi mã lệnh trong thành phần COM sẽ yêu cầu những thay đổi tương ứng đối với mã lệnh truy cập thành phần DCOM này.

Điểm khác biệt chính của SOA là cung cấp khả năng giao tiếp giữa các thành phần trong hệ thống sử dụng thông điệp (message) dựa trên các chuẩn giao tiếp đã được chuẩn hóa (HTTP, FTP, SMTP, ...). Chính nhờ đặc điểm này, hệ thống SOA trở nên độc lập với nền tảng (platform independent). Các dịch vụ hoạt động trên nền các nền tảng khác nhau vẫn có thể giao tiếp với nhau nhờ vào các giao diện giao tiếp đã được chuẩn hóa để cộng tác xử lý một tác vụ nào đó.

Trong môi trường SOA, các "nút" (node) trên một network sẽ giúp NSD tiếp cận dữ liệu theo phương thức chuẩn. SOA hoạt động trên một số nền như Java Enterprise System của Sun Microsystems, Indigo Application Server của Microsoft.

Như vậy, SOA cung cấp giải pháp để giải quyết các vấn đề tồn tại của các hệ thống hiện nay như: phức tạp, không linh hoạt và không ổn định. Một hệ thống triển khai theo mô hình SOA có khả năng dễ mở rộng, liên kết tốt. Đây chính là cơ sở và nền tảng cho việc tích hợp, tái sử dụng lại những tài nguyên hiện có.

Do đặc điểm chính của SOA là tách rời phần giao tiếp/gọi dịch vụ với phần thực hiện dịch vụ dẫn đến việc liên tưởng tới một công nghệ được đề cập nhiều gần đây là WebService (dịch vụ web). Dịch vụ web cho phép truy cập thông qua định nghĩa giao thức-và-giao tiếp. SOA và dịch vụ web thoạt trông có vẻ giống nhau nhưng chúng không phải là một.

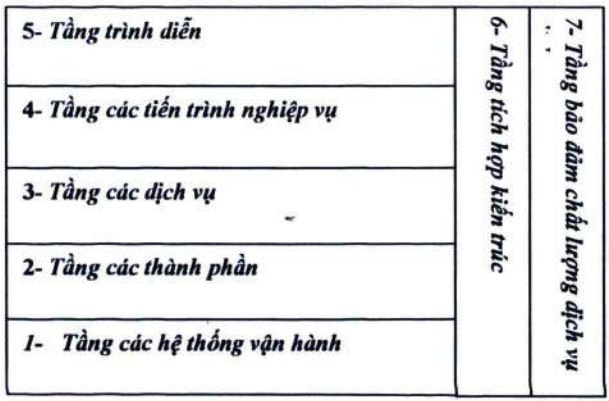
Theo định nghĩa thì dịch vụ web là đặc tả công nghệ còn SOA là triết lý thiết kế phần mềm. Dịch vụ web đưa ra giải pháp kỹ thuật để thực hiện SOA, nhưng SOA cũng có thể thực hiện với các giải pháp kỹ thuật khác không phải dịch vụ web (và không phải tất cả dịch vụ web đều có kiến trúc SOA). Tuy vậy, SOA và dịch vụ web có mối quan hệ tương hỗ: sự phổ biến của dịch vụ

web giúp thúc đẩy sự phát triển của SOA, và kiến trúc tốt của SOA sẽ giúp dịch vụ web thành công.

Kiến trúc hướng dịch vụ đang ngày càng được sử dụng trong ngữ cảnh của môi trường kinh doanh. Vấn đề quan trọng là xây dựng các *mẫu (pattern)* cho các phương pháp tiếp cận của quá trình tích hợp các dịch vụ. Ở đây, chúng ta cần quan tâm đến "khung công việc" (framework) cho các ngôn ngữ định hướng quá trình tích hợp các dịch vụ để mô tả đúng với thực tế. Một *nguyên mẫu (primitive pattern)* là cơ sở để mô tả chính xác các yếu tố mô hình biểu trưng cho các mẫu.

10.1.2. Các thành phần của một kiến trúc hướng dịch vụ

Các thành phần của một kiến trúc hướng dịch vụ là các tầng của SOA được thể hiện trên hình 10.1.



Hình 10.1. Các thành phần của một kiến trúc hướng dịch vụ

Danh mục dịch vụ dùng để mô tả các dịch vụ nghiệp vụ trong SOA, bao gồm một danh sách, phân loại và cấu trúc của các dịch vụ được xác định thông qua kỹ thuật phân tích và thiết kế hướng dịch vụ.

Các thành phần dùng để cung cấp các chức năng cuả dịch vụ. Các thành phần sử dụng dịch vụ, thành phần cung cấp dịch vụ và thành phần môi giới dịch vụ tại nơi đăng ký dịch vụ đều được định nghĩa và mô tả các dịch vụ công bố.

Còn các *tầng của SOA* dùng để biểu diễn vị trí của các thành phần và dịch vụ. Hình 10.1 mô tả các tầng của SOA. Với mỗi tầng, các quyết định thiết kế và kiến trúc được tiến hành.

Các tầng của SOA:

Tầng 1: *Tầng các hệ thống vận hành* chứa các hệ thống hoặc các ứng dụng hiện có, bao gồm các ứng dụng được đóng gói, các ứng dụng đang có, và các cài đặt hệ thống hướng đối tượng theo kiểu cũ cũng như các ứng dụng thu thập nghiệp vụ. Kiến trúc theo tầng của một kiến trúc hướng đối tượng có thể thúc đẩy các hệ thống hiện có, tích hợp chúng bằng cách sử dụng tích hợp hướng dịch vụ.

Tầng 2: *Tầng các thành phần* sử dụng các kỹ thuật và thiết kế hướng đối tượng trong thành phần truyền thống.

Tầng 3: *Tầng các dịch vụ* cung cấp các cơ chế để lấy các thành phần ở quy mô doanh nghiệp, các thành phần cho từng đơn vị doanh nghiệp cụ thể, và trong một số trường hợp là các thành phần của từng dự án và cung cấp các dịch vụ thông qua các giao diện của chúng. Trong tầng này, các giao diện được thể hiện ra ngoài thông qua các đặc tả dịch vụ, ở đó, các dịch vụ tồn tại một cách độc lập hoặc là tổng hợp của một số dịch vụ.

Tầng 4: *Tầng các tiến trình nghiệp vụ* là một sự tiến hoá của việc tổng hợp dịch vụ các luồng hoặc sắp đặt các nhiệm vụ được phân nhóm vào một luồng để hoạt động như một ứng dụng. Các ứng dụng này hỗ trợ các ca sử dụng các quy trình nghiệp vụ cụ thể. Ở đây, các công cụ tổng hợp luồng trực quan có thể được sử dụng để thiết kế luồng ứng dụng.

Tầng 5: *Tầng trình diễn* thường nằm ngoài phạm vi của một kiến trúc hướng dịch vụ.

Tầng 6: *Tầng tích hợp kiến trúc* cho phép sự tích hợp của các dịch vụ thông qua việc định tuyến tin cậy và thông minh, giao thức trung gian, và một số cơ chế chuyển đổi khác, thường được mô tả như là một tuyến dịch vụ doanh nghiệp (ESB- Enterprise Services Bus).

Tầng 7: *Tầng bảo đảm chất lượng dịch vụ* thông qua các cơ chế cảm biến và đáp ứng các công cụ kiểm soát các ứng dụng SOA.

10.1.3. Các tính chất của một hệ thống SOA

Hệ thống SOA có các tính chất đặc biệt sau [22]:

1- Kết nối lỏng (Loose coupling)

Vấn đề kết nối (coupling) ngầm định một số ràng buộc giữa các mô-đun với nhau. Có hai loại kết nối (coupling) là lỏng (loose) và chặt (tight). Các mô-đun có tính kết nối lỏng có một số ràng buộc được mô tả rõ ràng trong khi các mô-đun có tính kết nối chặt lại có nhiều ràng buộc không thể biết trước. Hầu như mọi kiến trúc phần mềm đều hướng đến tính kết nối lỏng giữa các mô-đun. Mức độ kết dính của mỗi hệ thống ảnh hưởng trực tiếp đến khả năng chỉnh sửa hệ thống của chính nó. Kết dính càng chặt bao nhiêu thì càng có nhiều thay đổi liên quan cần chỉnh sửa ở phía sử dụng dịch vụ mỗi khi có thay đổi nào đó xảy ra. Mức độ kết nối tăng dần khi bên sử dụng dịch vụ càng cần biết nhiều thông tin ngầm định của bên cung cấp dịch vụ để sử dụng dịch vụ được cung cấp. Nghĩa là nếu bên sử dụng dịch vụ biết vị trí và chi tiết định dạng dữ liệu của bên cung cấp dịch vụ thì quan hệ giữa hai bên càng chặt. Ngược lại, nếu bên sử dụng dịch vụ không cần biết mọi thông tin chi tiết của dịch vụ trước khi gọi nó thì quan hệ giữa hai bên càng có tính kết nối lỏng.

SOA hỗ trợ kết nối lỏng thông qua việc sử dụng hợp đồng và liên kết (contract and binding). Một người sử dụng truy vấn đến chỗ đăng ký dịch vụ (nơi lưu trữ và cung cấp thông tin dịch vụ-registry) để lấy thông tin về loại dịch vụ cần sử dụng. Nơi đăng ký sẽ trả về tất cả những dịch vụ thoả tiêu chuẩn tìm kiếm. Người dùng chỉ việc chọn dịch vụ mà mình cần, và thực thi phương thức trên đó theo mô tả dịch vụ nhận được từ nơi đăng ký. Bên sử dụng dịch vụ không cần phụ thuộc trực tiếp vào cài đặt của dịch vụ mà chỉ dựa trên hợp đồng mà dịch vụ đó hỗ trợ.

Tính kết nối lỏng giúp gỡ bỏ những ràng buộc điều khiển giữa những hệ thống đầu cuối. Mỗi hệ thống có thể tự quản lý độc lập nhằm tăng hiệu suất, khả năng mở rộng và khả năng đáp ứng cao. Những thay đổi cài đặt cũng được

che dấu đi. Kết nối lỏng đem đến sự độc lập giữa bên đăng ký và bên sử dụng nhưng nó đòi hỏi các giao diện phải theo chuẩn và cần một thành phần trung gian quản lý, trung chuyển yêu cầu giữa các hệ thống đầu cuối.

2- Sử dụng lại dịch vụ

Bởi vì các dịch vụ được cung cấp lên trên mạng và được đăng ký ở một nơi nhất định nên chúng dễ dàng được tìm thấy và tái sử dụng. Nếu một dịch vụ không có khả năng tái sử dụng, nó cũng không cần đến giao diện mô tả. Các dịch vụ có thể được tái sử dụng lại bằng cách kết hợp lại với nhau theo nhiều mục đích khác nhau. Tái sử dụng lại các dịch vụ còn giúp loại bỏ những thành phần trùng lặp và tăng độ vững chắc trong cài đặt, nó còn giúp đơn giản hoá việc quản trị. Những dịch vụ được dùng chung bởi tất cả các ứng dụng của một hệ thống SOA gọi là những dịch vụ cơ sở hạ tầng chia sẻ.

3- Sử dụng dịch vụ bất đồng bộ

Trong phương thức gọi dịch vụ bất đồng bộ, bên gọi gửi một thông điệp với đầy đủ thông tin ngữ cảnh tới bên nhận. Bên nhận xử lý thông tin và trả kết quả về thông qua một "kênh thông điệp", bên gọi không phải chờ cho đến khi thông điệp được xử lý xong. Khi sử dụng kết hợp thông điệp dạng "hạt thô" (coarse-grained) với một dịch vụ chuyển thông điệp, các yêu cầu dịch vụ có thể được đưa vào hàng đợi và xử lý với tốc độ tối ưu. Do bên gọi không phải chờ cho đến khi yêu cầu được xử lý xong và trả về nên không bị ảnh hưởng bởi việc xử lý trễ và lỗi khi thực thi các dịch vụ bất đồng bộ. Trên lý thuyết một hệ thống SOA có thể hỗ trợ gửi và nhận cả thông điệp đồng bộ và bất đồng bộ.

4- Quản lý các chính sách

Khi sử dụng các dịch vụ chia sẻ trên mạng, tùy theo mỗi ứng dụng sẽ có *một luật kết hợp riêng gọi là các chính sách*. Các chính sách cần được quản lý các áp dụng cho mỗi dịch vụ cả khi thiết kế lẫn khi trong thời gian thực thi. Việc này tăng khả năng tạo ra các dịch vụ có đặc tính tái sử dụng. Bởi vì các chính sách được thiết kế tách biệt, và tùy vào mỗi ứng dụng nên giảm tối đa các thay đổi phần mềm. Nếu không sử dụng các chính sách, các nhân viên phát triển phần mềm, nhóm điều hành và nhóm hỗ trợ phải làm việc với nhau trong

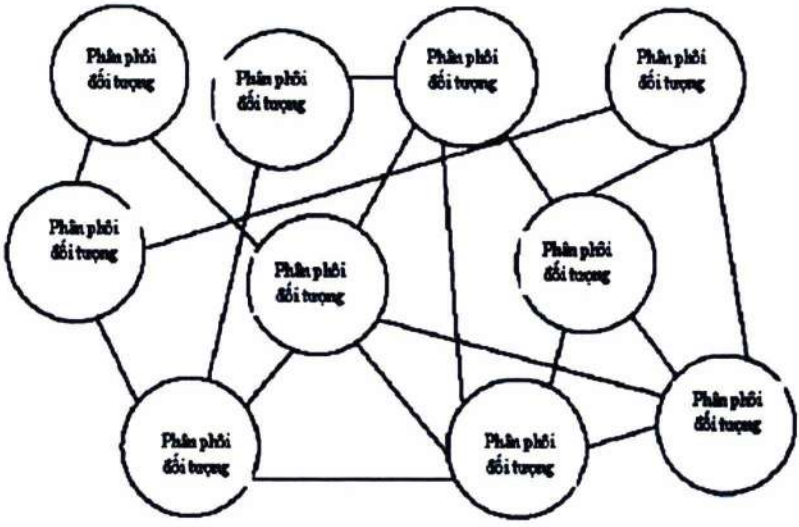
suốt thời gian phát triển để cài đặt và kiểm tra những chính sách. Ngược lại, nếu sử dụng chính sách, những nhân viên phát triển phần mềm chỉ cần tập trung vào quy trình nghiệp vụ trong khi nhóm điều hành và nhóm hỗ trợ tập trung vào các luật kết hợp.

5- Độ thô mịn (Coarse granularity)

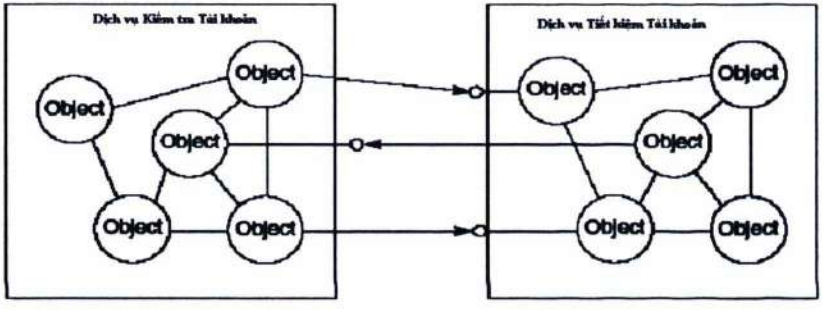
Khái niệm độ mịn trong dịch vụ có thể hiểu theo hai cách. Đầu tiên, nó được hiểu trong phạm vi toàn bộ kiến trúc cài đặt của dịch vụ. Thứ hai, nó được hiểu trong phạm vi từng phương thức của từng giao diện triển khai. Mức độ mịn cũng được hiểu ở mức tương đối. Chẳng hạn, nếu một dịch vụ cài đặt tất cả chức năng của một hệ thống ngân hàng, chúng ta xem nó là *hạt thô (coarse-grained)*. Nếu nó hỗ trợ chỉ chức năng kiểm tra thẻ tín dụng, chúng ta lại xem nó là *hạt mịn (fine-grained)*. Trước khi có kiến trúc thành tố và dịch vụ, các hệ thống phân tán chủ yếu dựa trên ý tưởng phân tán đối tượng. Những hệ thống phân tán đối tượng chứa bên trong nó nhiều đối tượng hạt mịn trao đổi thông tin với nhau qua mạng. Mỗi đối tượng có những ràng buộc với nhiều đối tượng khác bên trong hệ thống. Do truy cập đến một đối tượng phải qua nhiều trung gian mà hiệu quả đạt được không cao nên khuynh hướng thiết kế hệ thống phân tán đối tượng đang dần chuyển sang thiết kế các giao diện hạt thô (coarser-grained interface).

Hình 10.2 minh họa một hệ thống phân tán đối tượng với nhiều mối liên kết cùng với kích thước và độ phức tạp của hệ thống ngày càng tăng, những ràng buộc này trở nên ngày càng khó quản lý. Hiệu suất cũng giảm tương ứng số lượng các kết nối trung gian. Khả năng bảo trì cũng giảm khi số lượng ràng buộc giữa những đối tượng ngày một tăng. Khi một đối tượng cần được thay đổi ở giao diện, nó có thể ảnh hưởng đến một lượng lớn những đối tượng phân tán khác. Nhân viên phát triển phải biên dịch và triển khai lại toàn bộ đối tượng bị thay đổi và những đối tượng liên quan với chúng.

Một hệ thống dựa trên quản lý các truy cập đến đối tượng bên trong dịch vụ thông qua một số giao diện hạt thô như hình 10.3. Một dịch vụ có thể được cài đặt như một tập những đối tượng hạt mịn nhưng bản thân những đối tượng đó lại được sử dụng trực tiếp qua mạng.



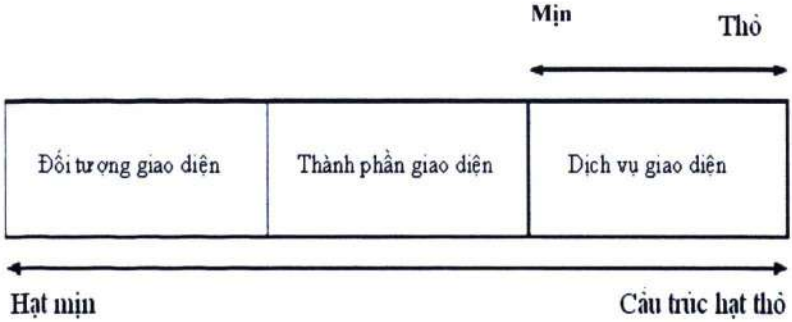
Hình 10.2. Các đối tượng hạt mịn (fine-grained) [22]



Hình 10.3. Minh họa các đối tượng hạt thô [22]

Một dịch vụ được cài đặt như những đối tượng có một hoặc nhiều *đối tượng hạt thô (coarse-grained)* hoạt động như những mặt tiền (facades) phân tán thì những đối tượng này lại có thể được sử dụng qua mạng và cho phép truy cập đến các đối tượng sâu bên trong. Tuy nhiên các đối tượng bên trong dịch vụ đó bây giờ sẽ trao đổi trực tiếp với nhau ở trong cùng một máy chứ không phải trên mạng.

Mặc dù những dịch vụ nói chung hỗ trợ giao diện thô hơn các hệ thống phân tán đối tượng và các hệ thống hướng thành tố, độ "thô" vẫn hàm chứa bên trong nó chứa một mức độ mịn *(granularity)* nào đó, như hình 10.4.



Hình 10.4. Các mức độ mịn [22]

6- Khả năng cộng tác (Interoperability)

Kiến trúc hướng dịch vụ nhấn mạnh đến khả năng cộng tác, khả năng mà các hệ thống có thể giao tiếp với nhau trên nhiều nền tảng và ngôn ngữ khác nhau. Mỗi dịch vụ cung cấp một giao diện có thể được gọi thông qua một dạng kết nối. Một kết nối gọi là dữ liệu khả kết (interoperable) chứa bên trong nó một giao thức và một định dạng dữ liệu mà mỗi khách hàng kết nối đến nó đều hiểu. Khả năng cộng tác được thực hiện bằng cách hỗ trợ các giao thức và định dạng dữ liệu chuẩn của dịch vụ và các khách hàng. Kỹ thuật này đạt được bằng cách ánh xạ mỗi tính chất và ngôn ngữ qua một đặc tả trung gian. Đặc tả trung gian sẽ chịu trách nhiệm ánh xạ giữa định dạng của dữ liệu khả kết đến định dạng dữ liệu tùy thuộc vào nền tảng hệ thống. Chẳng hạn, WebService là một đặc tả trung gian cho giao tiếp giữa các hệ thống, JAX-RPC và JAXM chuyển đối tượng dạng Java thành SOAP.

7- Tự động dò tìm và ràng buộc động

SOA hỗ trợ khái niệm truy tìm dịch vụ (Service discovery). Một NSD cần đến một dịch vụ nào đó có thể tìm kiếm dịch vụ dựa trên một số tiêu chuẩn khi cần. NSD chỉ cần hỏi một nơi đăng ký dịch vụ về dịch vụ nào thoả mãn yêu cầu tìm kiếm. Giả sử, một người dùng (consumer) yêu cầu một nơi đăng ký tìm

tất cả các dịch vụ có khả năng kiểm tra thẻ tín dụng. Nơi đăng ký trả về một tập các đầu vào (entry) thoả mãn yêu cầu.

Các đầu vào chứa thông tin về dịch vụ, bao gồm cả phí giao dịch. Bên sử dụng sẽ chọn một dịch vụ có phí giao dịch thấp nhất trong danh sách các dịch vụ trả về, kết nối đến nơi đăng ký dịch vụ đó dựa trên thông tin đầu vào của nơi đăng ký để sử dụng dịch vụ kiểm tra thẻ tín dụng.

Trong phần mô tả dịch vụ kèm theo đã có tất cả các tham số cần thiết dùng để thực thi dịch vụ, bên sử dụng chỉ cần định dạng dữ liệu yêu cầu đúng theo mô tả cung cấp và gửi đi. Nơi đăng ký dịch vụ sẽ thực thi kiểm trả thẻ tín dụng và trả về một thông điệp có định dạng đúng như trong phần mô tả dịch vụ. Mối ràng buộc duy nhất giữa bên cung cấp và bên sử dụng là bản hợp đồng được cung cấp bởi nơi đăng ký dịch vụ trung gian. Mối ràng buộc này là ràng buộc trong thời gian chạy chứ không phải ràng buộc trong lúc biên dịch. Tất cả thông tin cần thiết về dịch vụ được lấy về và sử dụng trong khi chạy.

Minh họa trên cho thấy cách bên sử dụng gọi dịch vụ một cách "động". Đây là một thế mạnh của SOA. Với SOA, bên sử dụng dịch vụ không cần biết định dạng của thông điệp yêu cầu và thông điệp trả về, cũng như địa chỉ dịch vụ cho đến khi cần.

8- Tự hồi phục (self-healing)

Với kích cỡ và độ phức tạp của những ứng dụng phân tán ngày nay, khả năng phục hồi của một hệ thống sau khi bị lỗi trở thành một yếu tố quan trọng. Một hệ thống tự hồi phục là một hệ thống có khả năng tự hồi phục sau khi bị lỗi mà không cần sự can thiệp của con người.

Độ tin cậy (reliability) là mức độ đo khả năng một hệ thống xử lý tốt như thế nào trong tình trạng hỗn loạn. Trong kiến trúc hướng dịch vụ, các dịch vụ luôn có thể hoạt động hay ngừng bất kỳ lúc nào, nhất là đối với những ứng dụng tổng hợp từ nhiều những dịch vụ của nhiều tổ chức khác nhau. Độ tin cậy phụ thuộc vào khả năng phục hồi của phần cứng sau khi bị lỗi. Hạ tầng mạng phải cho phép các kết nối động từ nhiều hệ thống khác nhau kết nối đến trong khi chạy. Một khía cạnh khác ảnh hưởng đến độ tin cậy là kiến trúc mà dựa trên đó ứng dụng được xây dựng. Một kiến trúc hỗ trợ kết nối và thực thi động

khi chạy sẽ có khả năng tự phục hồi hơn một hệ thống không hỗ trợ những tính năng trên.

Do những hệ thống dựa trên dịch vụ yêu cầu sự tách biệt giữa dịch vụ và cài đặt nên có thể có nhiều cài đặt khác nhau cho cùng một giao diện. Nếu một thể hiện dịch vụ nào đó không hoạt động thì một thể hiện khác vẫn có thể hoàn tất giao dịch cho khách hàng mà không bị ảnh hưởng gì. Khả năng này chỉ có được khi khách hàng chỉ tương tác với giao diện của dịch vụ chứ không tương tác trực tiếp cài đặt của dịch vụ. Đây là một trong những tính chất cơ bản của các hệ thống hướng dịch vụ.

10.2. Công nghệ dịch vụ

10.2.1. Nhận dạng kiểu dịch vụ thích hợp

Dịch vụ là khái niệm then chốt trong SOA. Có thể hiểu dịch vụ như là một chức năng/ hàm chức năng (mô-đun phần mềm) thực hiện qui trình nghiệp vụ nào đó. Nó được xác định rõ ràng, tự chứa, và không phụ thuộc vào ngữ cảnh hoặc tình trạng của các dịch vụ khác.

Một cách cơ bản, SOA là tập hợp các dịch vụ kết nối "mềm dẻo" với nhau (nghĩa là một ứng dụng có thể "nói chuyện" với một ứng dụng khác mà không cần biết các chi tiết kỹ thuật bên trong), có giao tiếp (dùng để gọi hàm dịch vụ) được định nghĩa rõ ràng và độc lập với nền tảng hệ thống, và có thể tái sử dụng. SOA là cấp độ cao hơn của phát triển ứng dụng, chú trọng đến qui trình nghiệp vụ và dùng giao tiếp chuẩn để giúp che đi sự phức tạp kỹ thuật bên dưới.

Các dịch vụ trong SOA có đặc điểm sau:

1- Các dịch vụ là có thể tìm kiếm được;

2- Các dịch vụ có tính liên thông;

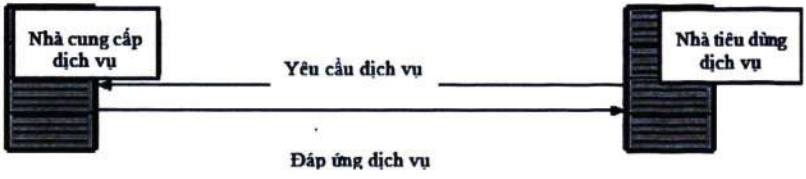
3- Các dịch vụ không được gắn kết chặt chẽ với nhau;

4- Các dịch vụ là phức hợp, bao gồm nhiều thành phần, được đóng gói ở mức cao;

5- Các dịch vụ trong suốt về vị trí;

6- Các dịch vụ có khả năng tự hàn gắn.

Hình 10.5 minh họa một SOA cơ bản. Nó cho thấy một người tiêu dùng dịch vụ ở bên phải gửi một thông điệp yêu cầu dịch vụ cho một nhà cung cấp dịch vụ ở bên trái. Các nhà cung cấp dịch vụ có thông điệp phản hồi cho người tiêu dùng dịch vụ. Các kết nối yêu cầu và phản hồi được xác định theo một cách nào đó sao cho cả người tiêu dùng dịch vụ và nhà cung cấp dịch vụ hiểu được. Một nhà cung cấp dịch vụ cũng có thể là một người tiêu dùng dịch vụ.



Hình 10.5. Một SOA cơ bản [9]

Các dịch vụ nhằm cung cấp các mô tả dịch vụ và liên lạc theo dạng thông điệp sẽ hình thành nên một kiến trúc cơ sở. Từ đó, kiến trúc này lại sinh ra các kiến trúc phân tán nhằm hỗ trợ thông điệp và tách giao diện khỏi logic xử lý. Vấn đề là ba thành phần cốt lõi (các dịch vụ, mô tả, và các thông điệp) sẽ được thiết kế như thế nào. Đây chính là phần mà hướng dịch vụ quan tâm.

Chúng ta cần chú trọng vào một số khía cạnh cốt lõi về dịch vụ có tính nguyên tắc sau:

- Các dịch vụ được ghép nối lỏng lẻo duy trì một quan hệ phụ thuộc cực tiểu và chỉ yêu cầu chúng giữ được nhận biết các dịch vụ khác.

- Các dịch vụ làm nhiệm vụ hợp đồng dịch vụ phải gắn liền với các thoả thuận liên lạc, như được xác định chung bằng một hoặc nhiều hơn các mô tả dịch vụ và các tài liệu liên quan.

- Các dịch vụ tự trị có điều khiển thông qua logic bao bọc chúng.

- Cần làm rõ cái gì sẽ được mô tả trong hợp đồng dịch vụ, trong các logic ẩn dịch vụ từ thế giới bên ngoài.

- Logic tái sử dụng cần được chia nhỏ thành các dịch vụ con để tăng cường tái sử dụng.

- Các sưu tập dịch vụ có thể được hợp tác và lắp ráp thành dạng các dịch vụ tổ hợp.

- Các dịch vụ trạng thái cực tiểu hoá duy trì đặc tả thông tin cho mỗi hoạt động.

- Các dịch vụ khám phá được thiết kế để mô tả bề ngoài sao cho chúng có thể tìm thấy và truy cập qua các cơ chế khám phá.

Như vậy, khi nói đến SOA, người ta ngụ ý một loạt dịch vụ trên Internet hoặc Intranet có sử dụng các tiện ích web. SOA về bản chất là một tập hợp các dịch vụ có liên hệ với nhau, có thể 2-3 dịch vụ cùng phối hợp thực hiện một hoạt động nào đó. Khi những dịch vụ này tích hợp với nhau thì sẽ cần đến các phương tiện kết nối, trong đó có các tiêu chuẩn như XML (ngôn ngữ đánh dấu mở rộng), giao thức HTTP, SOAP (giao thức truy cập đối tượng đơn giản), WSDL (ngôn ngữ mô tả dịch vụ web), UDDI (tích hợp, phát hiện, mô tả tổng hợp).

10.2.2. Thiết kế dịch vụ

Thiết kế SOA tách riêng phần thực hiện dịch vụ (phần mềm) với giao tiếp gọi dịch vụ. Điều này tạo nên một giao tiếp nhất quán cho ứng dụng khách (Client) sử dụng dịch vụ bất chấp công nghệ thực hiện dịch vụ.

Thay vì xây dựng các ứng dụng đơn lẻ và đồ sộ, nhà phát triển sẽ xây dựng các dịch vụ tinh gọn có thể triển khai và tái sử dụng trong toàn bộ quy trình nghiệp vụ. Điều này cho phép tái sử dụng phần mềm tốt hơn, cũng như tăng sự linh hoạt vì nhà phát triển có thể cải tiến dịch vụ mà không làm ảnh hưởng đến ứng dụng Client sử dụng dịch vụ.

Trong SOA có ba đối tượng chính, minh họa trong hình 10.6.

Hình 10.6. Sơ đồ cộng tác trong SOA [22]

Nhà cung cấp dịch vụ (service provider) sẽ đăng ký thông tin về dịch vụ mà mình có thể cung cấp (các chức năng có thể cung cấp, khả năng của hệ thống, giá cả dịch vụ, ...) ở trụ sở đăng ký dịch vụ (Service registry). Người sử dụng (Service consumer) khi có nhu cầu về một dịch vụ nào đó sẽ tìm kiếm thông tin tại trụ sở đăng ký dịch vụ. Ngoài chức năng hỗ trợ tìm kiếm, trụ sở đăng ký dịch vụ còn có thể xếp hạng các nhà cung cấp dịch vụ dựa trên các tiêu chí về chất lượng dịch vụ, bầu chọn từ các khách hàng đã sử dụng dịch vụ,... Khi đã xác định được nhà cung cấp dịch vụ mong muốn, NSD thiết lập kênh giao tiếp trực tiếp với nhà cung cấp dịch vụ nhằm sử dụng dịch vụ hoặc tiến hành thương lượng thêm (về mặt giá cả, tài nguyên sử dụng,...).

Vì ưu điểm quan trọng nhất của SOA là khả năng kết nối "mềm dẻo" (nhờ sự chuẩn hóa giao tiếp) *và tái sử dụng* nên các dịch vụ có thể được sử dụng với trình Client chạy trên nền tảng bất kỳ và được viết với ngôn ngữ bất kỳ (chẳng hạn, ứng dụng Java có thể liên kết với dịch vụ viết trên nền .NET và ngược lại).

Việc thiết kế dịch vụ phải tuân theo 4 nguyên tắc chính sau đây:

Nguyên tắc1. Sự phân định ranh giới rạch ròi giữa các dịch vụ

Các dịch vụ thực hiện quá trình tương tác chủ yếu thông qua thành phần giao tiếp. Thành phần giao tiếp này sẽ qui định về những định dạng thông điệp sử dụng trong quá trình trao đổi: thông điệp nào sẽ được chấp nhận và thông điệp nào sẽ không được xử lý. Và đây chính là cách duy nhất để các đối tượng bên ngoài có thể truy cập thông tin và chức năng của dịch vụ. Ta chỉ cần gửi các thông điệp theo các định dạng đã được định nghĩa trước mà không cần phải quan tâm đến cách xử lý của dịch vụ như thế nào (môi trường thực thi, ngôn ngữ lập trình...). Điều này đạt được do sự tách biệt giữa thành phần giao tiếp và thành phần xử lý trong kiến trúc của dịch vụ.

Nguyên tắc 2. Các dịch vụ tự hoạt động

Các dịch vụ cần phải được triển khai và hoạt động như những thực thể độc lập mà không lệ thuộc vào một dịch vụ khác. Dịch vụ phải có tính bền vững cao, nghĩa là nó sẽ không bị sụp đổ khi có sự cố. Để thực hiện điều này, dịch vụ cần duy trì đầy đủ thông tin cần thiết cho quá trình hoạt động của mình

để có thể tiếp tục hoạt động trong trường hợp một dịch vụ cộng tác bị hỏng; và để tránh các cuộc tấn công từ bên ngoài (như gửi thông điệp lỗi, hay gửi thông điệp ồ ạt) bằng cách sử dụng các kỹ thuật về an toàn, bảo mật...

Nguyên tắc 3. Các dịch vụ chia sẻ lược đồ dữ liệu

Các dịch vụ có trách nhiệm cung cấp thành phần giao tiếp của nó (interface) ra bên ngoài, và hỗ trợ chia sẻ các cấu trúc thông tin, các ràng buộc dữ liệu thông qua các lược đồ dữ liệu (schema) chuẩn (độc lập ngôn ngữ, độc lập hệ nền). Như thế hệ thống sẽ có tính liên kết và khả năng dễ mở rộng.

Nguyên tắc 4. Tính tương thích của dịch vụ dựa trên chính sách

Điều này nghĩa là, một dịch vụ khi muốn tương tác với một dịch vụ khác thì phải thỏa mãn các chính sách (policy) và yêu cầu (requirements) của dịch vụ đó như là mã hóa, bảo mật... Để thực hiện điều này, mỗi dịch vụ cần phải cung cấp công khai các yêu cầu, chính sách đó.

Kiến trúc hướng dịch vụ SOA quan tâm đến 2 vấn đề rất quan trọng:

1-Mô-đun: Tách vấn đề lớn thành nhiều vấn đề nhỏ;

2-Đóng gói: Che đi dữ liệu và logic trong từng mô-đun (hay "hộp đen") đối với truy cập từ ngoài.

Một thiết kế kiến trúc phù hợp với khái niệm của SOA cần có những tính chất sau:

- Một dịch vụ là một đơn vị phần mềm gồm các hoạt động nghiệp vụ có tính chứa đựng và mức độ đóng gói cao (coarse-grained).

- Một dịch vụ có thể dùng lại được, cho phép có thể xây dựng được một dịch vụ mới từ các dịch vụ hiện có. Do đó, việc quan sát các hàm ý có thể có của các thuộc tính phi chức năng như tính giao dịch là rất quan trọng.

- Một giao diện dịch vụ là một điểm cuối mạng (Network Endpoint) bảo đảm tính độc lập và trong suốt về vị trí.

- Một dịch vụ cần có khả năng được phát hiện ra một cách công khai bằng cách sử dụng một trụ sở đăng ký dịch vụ nhằm cho phép các liên kết động tới dịch vụ.

- Một dịch vụ cần bảo đảm tính liên thông bằng cách hỗ trợ các giao thức truyền thông được chuẩn hoá và các định dạng dữ liệu rõ ràng.

- Các đặc điểm trên bảo đảm cho một kiến trúc hướng dịch vụ khả năng gắn kết lỏng lẻo của các dịch vụ phân tán và có tính mô-đun bằng cách sử dụng các giao ước dịch vụ để mô tả các định dạng thông điệp cần thiết.

10.2.3. Các hoạt động phát triển dịch vụ

Các hoạt động phát triển dịch vụ trong hệ thống bao gồm [28]:

1-Xuất bản dịch vụ: để có thể truy cập được, mô tả dịch vụ phải được xuất bản để nó có thể được tìm thấy và triệu gọi bởi một người dùng dịch vụ.

2-Tìm kiếm dịch vụ: một người yêu cầu dịch vụ định vị một dịch vụ bằng cách yêu cầu nơi đăng ký dịch vụ một dịch vụ phù hợp với các tiêu chí đặt ra.

3-Liên kết và thực thi dịch vụ: sau khi nhận được mô tả dịch vụ, người dùng sẽ gọi dịch vụ theo các thông tin mô tả.

Trong hình 10.6, chúng ta hiểu rằng đầu tiên thành phần cung cấp dịch vụ sẽ đăng ký dịch vụ của mình với một UDDI Service (thông tin đăng ký được đặc tả theo định dạng gọi là ngôn ngữ đặc tả dịch vụ - WSDL), khi đã đăng ký rồi thì thành phần sử dụng dịch vụ nếu muốn sử dụng thì sẽ truy xuất UDDI Service để lấy thông tin về vị trí dịch vụ hiện đang đặt trên địa chỉ nào, sau đó thực hiện yêu cầu mong muốn và sau khi chạy dịch vụ đó sẽ trả kết quả về cho phía sử dụng dịch vụ.

10.2.4. Một số công nghệ tiêu biểu xây dựng kiến trúc hệ thống theo hướng dịch vụ

Trong chương 6, chúng ta đã nghiên cứu đặc tả kiến trúc hệ thống được thiết kế theo hướng dịch vụ. Ở đây, chúng ta chỉ thêm phần giới thiệu một số công nghệ tiêu biểu để tạo dựng kiến trúc hệ thống theo hướng dịch vụ:

10.2.4.1. Sun JINI

Công nghệ Jini cho phép xây dựng một hệ thống là một mạng của các dịch vụ. Các dịch vụ có thể được thêm vào và xoá bỏ khỏi mạng, và người

dùng mới có thể tìm kiếm các dịch vụ hiện có. Dịch vụ được dựa trên các giao diện đã được viết trong ngôn ngữ lập trình Java. Nó không quan tâm đến việc dịch vụ cài bằng phần cứng hay phần mềm.

Đối tượng dịch vụ mà người dùng tải về được cung cấp bởi các thành phần cung cấp dịch vụ. Client chỉ cần biết rằng họ đang làm việc với một cài đặt của một giao diện được viết bằng ngôn ngữ lập trình Java. Việc thiết kế dựa trên các giao diện dịch vụ cho phép xây dựng các hệ thống với tính sẵn dùng cao.

Một thành phần có thể sử dụng bất kỳ dịch vụ nào phù hợp với giao diện, thay vì được cấu hình tĩnh để giao tiếp với một thành phần nhất định nào đó.

Công nghệ Jini được xây dựng phía trên công nghệ Java (hình 10.7). Phương thức triệu gọi từ xa của Java (RMI) cung cấp cơ chế thu rác từ xa của các đối tượng từ xa và khả năng chuyển trạng thái của đối tượng cũng như mã đối tượng qua mạng.

Hình 10.7. Mô hình kiến trúc Jini [28]

Kiến trúc Jini bao gồm:

1-Dịch vụ tra cứu (Looup Services):

Dịch vụ tra cứu lưu trữ các dịch vụ Jini và cung cấp các uỷ nhiệm để truyền thông với dịch vụ, bản thân nó cũng là một dịch vụ Jini.

2-Dịch vụ Jini(Jini services):

Dịch vụ Jini được đăng ký với dịch vụ tra cứu và có khả năng được triệu gọi thông qua giao diện công khai của mình (giao diện này được định nghĩa thông qua một giao diện từ xa). Hệ thống nền tảng truyền thông của Jini là RMI.

3-Thành phần sử dụng dịch vụ Jini (Jini Client):

Thành phần sử dụng dịch vụ Jini là một phần mềm yêu cầu đối tượng uỷ nhiệm từ dịch vụ tra cứu để gọi dịch vụ Jini.

Jini có một số các dịch vụ được xây dựng sẵn, bao gồm:

1-Dịch vụ tìm kiếm tra cứu (Lookup Discovery Services):

2-Dịch vụ tìm kiếm tra cứu thông báo cho các thành phần sử dụng dịch vụ về các thay đổi trong mạng Jini. Các dịch vụ có thể kết hợp hoặc tách ra khỏi mạng bất kỳ thời điểm nào.

3-Dịch vụ tái tạo ràng buộc(Lease Renewal Services):

Khái niệm tái tạo ràng buộc hỗ trợ mạng dịch vụ Jini tính năng tự hàn gắn và khắc phục lỗi. Dịch vụ phải tái tạo ràng buộc không được tái tạo, dịch vụ sẽ là một ứng viên bị loại bỏ khỏi mạng lưới dịch vụ. Trách nhiệm của những người quản trị trong lĩnh vực này được giảm tới mức tối thiểu nhờ tính năng tự hàn gắn của hệ thống.

4-Dịch vụ giao dịch (Transactor Services):

Dịch vụ cho phép sử dụng các giao dịch trong một hệ thống phân tán. Thông thường, các tổ chức sử dụng CSDL để tạo các hệ thống giao dịch. Dịch vụ giao dịch của Jini đưa tính năng giao dịch của CSDL lên mạng. Các dịch vụ có thể tham gia vào các giao dịch để bảo đảm các thuộc tính ACID (Atomicity-Consistency- Isolation- Durability) gắn liền với giao dịch.

5-Dịch vụ hộp thư sự kiện (Event MailBox Services):

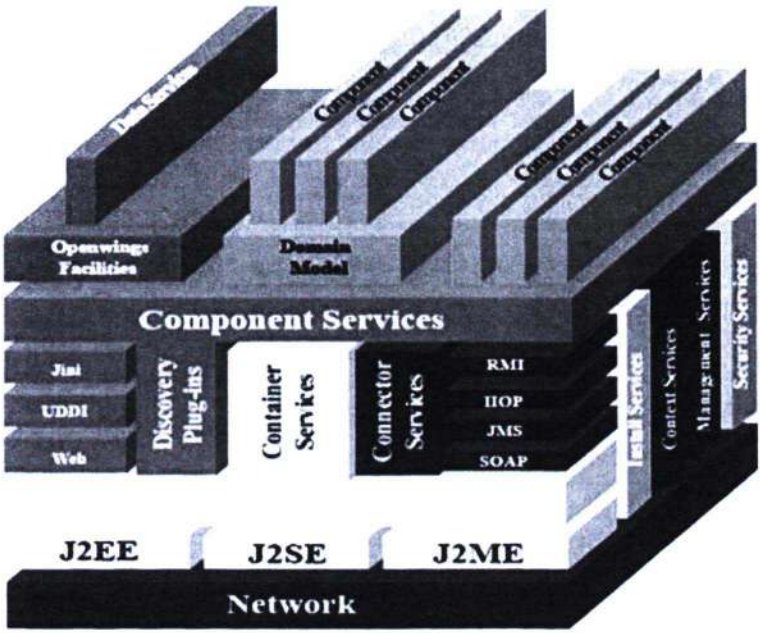
Các thay đổi trong mạng dịch vụ Jini được truyền đi trong hệ thống bằng cách sử dụng các sự kiện phân tán. dịch vụ hộp thư sự kiện hỗ trợ tính năng thông báo các sự sự kiện cho các dịch vụ ngay cả khi dịch vụ không được kích hoạt tại thời điểm hiện tại.

10.2.4.2. Openwings

Openwings là một khung kiến trúc hướng dịch vụ cho việc xây dựng các hệ thống các siêu hệ thống (hệ thống của hệ thống). Mặc dù không bị ràng buộc cụ thể với Jini, nó cho phép xây dựng dựa trên các khái niệm của Java và Jini để cung cấp một giải pháp hoàn thiện hơn.

Openwings có hàng loạt các dịch vụ cốt lõi hỗ trợ tính toán hướng đối tượng:

1-Component Services (Các dịch vụ thành phần): cung cấp các kỹ thuật cho việc đưa ra công bố một dịch vụ trong hệ thống bảo đảm cho quá trình tìm kiếm phát hiện (discover).



Hình 10.8. Mô hình khung của Openwings [28]

2-Install services (Các dịch vụ cài đặt): là một thành phần dịch vụ của framework cho phép thực hiện cài đặt các thành phần mới vào hệ thống. Để cài đặt được thì các thành phần này cần phải thông qua các quy trình chuẩn mà Openwings đề xuất.

3-Context Services (Các dịch vụ ngữ cảnh): là các thành phần cho hỗ trợ việc kết hợp các dịch vụ trong môi trường hệ thống để có được một dịch vụ lớn hơn. Đồng thời còn hỗ trợ việc kết hợp các dịch vụ được chia sẻ trong một mạng WAN.

4-Management Services (các dịch vụ quản lý): cung cấp các phương thức cơ bản hỗ trợ việc quản lý các thành phần và dịch vụ trong hệ thống. Với các hỗ trợ từ dạng quản lý hệ thống qua can thiệp bằng tay hay thông qua việc cài đặt các cơ chế quản lý tự động. Các hỗ trợ này được đóng gói trong Mbean.

5-Security Services (các dịch vụ kết bảo mật): cung cấp các phương thức cho việc mã hoá, truyền tải dữ liệu trong hệ thống, các hỗ trợ cơ bản cho việc cấp quyền, thẩm định quyền, đảm bảo an toàn khi truyền tin, tính toàn vẹn, khả năng phát hiện tấn công và đáp trả các tấn công vào hệ thống.

6-Connector Services (các dịch vụ kết nối): cung cấp các kỹ thuật cho việc giao tiếp giữa các thành phần hay dịch vụ trong hệ thống. Hỗ trợ trực tiếp cho các kỹ thuật giao tiếp thông qua một đối tượng chuyển tiếp trung gian như CORBA, RMI, giao tiếp từ một giao diện dịch vụ được định hướng trước theo các kỹ thuật chuyển tiếp trung gian chuẩn với các cơ chế động ngay tại thời điểm diễn ra giao tiếp tuỳ theo yêu cầu sử dụng.

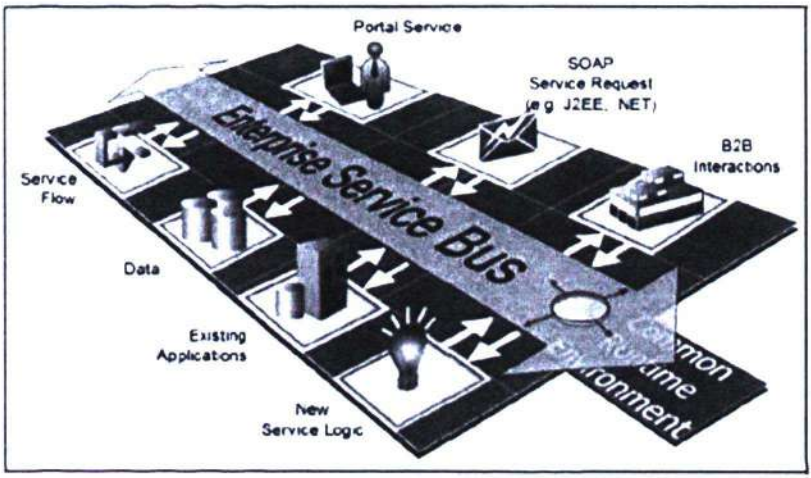
7-Container Services (các dịch vụ chứa đựng): cung cấp môi trường tương ứng cho việc thực hiện các dịch vụ trong mô hình Openwings. Đây là một thành phần quan trọng trong việc tạo nên khả năng vận hành động không phụ thuộc vào nền tảng hay môi trường vận hành phân tán của hệ thống.

10.2.4.3. Dịch vụ web

Mặc dù các khái niệm nền tảng cho kiến trúc hướng dịch vụ đã được thiết lập từ trước khi dịch vụ mạng xuất hiện nhưng dịch vụ mạng vẫn đóng một vai trò quan trọng trong một kiến trúc hướng dịch vụ. Đó là bởi vì dịch vụ mạng được xây dựng trên một tập các giao thức được biết tới nhiều và độc lập về nền tảng. Các giao thức này bao gồm HTTP, XML, UDDI, và SOAP. Chính sự kết hợp của các giao thức này đã làm dịch vụ mạng đáp ứng được các yêu cầu của chính kiến trúc hướng dịch vụ: SOA yêu cầu dịch vụ phải được phát hiện và triệu gọi động, yêu cầu này được thoả mãn bằng UDDI, WSDL, và SOAP; SOA yêu cầu dịch vụ có một giao ước giao diện độc lập nền tảng, yêu cầu này được thoả mãn bởi XML; SOA nhấn mạnh tới tính liên thông, yêu cầu này được thoả mãn bởi HTTP. Đây là lý do tại sao các dịch vụ mạng mang lại có vai trò trung tâm trong kiến trúc hướng dịch vụ.

10.2.4.4. Enterprise Services Bus (ESB)

Các công nghệ dựa trên nền dịch vụ mạng ngày càng được ứng dụng rộng rãi trong phát triển và tích hợp ứng dụng trong các tổ chức. Một trong những vấn đề nổi lên hiện nay là việc tìm kiếm các phương pháp hiệu quả hơn trong việc thiết kế, phát triển và triển khai dịch vụ mạng dựa trên các hệ thống kinh doanh; quan trọng hơn nữa là việc chuyển kiểu truyền thông dịch vụ mạng điểm tới điểm tới các ứng dụng lớn hơn của các công nghệ này thành các quy trình kinh doanh ở mức xí nghiệp. Trong bối cảnh này, mô hình ESB đang xuất hiện như một bước tiến mới trong sự phát triển của dịch vụ mạng và kiến trúc hướng đối tượng.



Hình 10.9. Mô hình ESB [28]

Dịch vụ mạng cơ bản (SOAP/HTTP điểm tới điểm) cung cấp một nền tảng vững trắc cho việc cài đặt một kiến trúc hướng dịch vụ, nhưng có những vấn đề ảnh hưởng tới tính mềm dẻo và khả năng bảo trì của chúng trong các kiến trúc ở quy mô xí nghiệp như:

-Thứ nhất, bản chất điểm tới điểm của dịch vụ mạng cơ bản có nghĩa là người dùng dịch vụ thường cần phải được sửa đổi bất kỳ khi nào giao diện người cung cấp dịch vụ thay đổi. Điều này không thành vấn đề trên quy mô

nhỏ, nhưng trong các xí nghiệp lớn chúng ta sẽ phải phải thay đổi đối với các trình khách đã có.

-Thứ hai, chúng ta có thể kết thúc bằng một kiến trúc dễ đổ vỡ và không linh hoạt khi một số dung lượng lớn người dùng dịch vụ và người cung cấp dịch vụ liên lạc với nhau sử dụng các kết nối kiểu điểm tới điểm hỗn loạn.

-Thứ ba, dịch vụ mạng cơ bản đòi hỏi mỗi người dùng phải có một bộ điều hợp giao thức thích hợp với mỗi nhà cung cấp dịch vụ. Việc phải triển khai nhiều bộ điều hợp giao thức trên nhiều ứng dụng khách làm tăng giá thành và chi phí bảo trì.

Người ta hy vọng rằng cách tiếp cận ESB sẽ giải quyết được những vấn đề này.

ESB không phải là một sản phẩm, nhưng là một thực tiễn kiến trúc tốt nhất cho việc cài đặt một kiến trúc hướng dịch vụ. Nó thiết lập một tuyến đường thông điệp lớp xí nghiệp kết hợp hạ tầng thông điệp với sự chuyển đổi thông điệp và định tuyến dựa vào nội dung trong một tầng tích hợp logic giữa những người dùng và người cung cấp dịch vụ.

Mục đích của ESB là cung cấp một sự trừu tượng hoá về các nguồn tài nguyên của doanh nghiệp, cho phép các nghiệp vụ của doanh nghiệp có thể được phát triển và quản lý độc lập với hạ tầng, mạng và có sự có mặt của các dịch vụ doanh nghiệp khác. Các nguồn tài nguyên trong ESB được mô hình như những dịch vụ có một hay nhiều thao tác.

Cài đặt một ESB đòi hỏi một tập hợp được tích hợp của các dịch vụ phần giữa *(middleware)*, được cung cấp bởi một ESB, hỗ trợ các kiểu kiến trúc như sau:

1-Các kiến trúc hướng dịch vụ: ở đây, các ứng dụng phân tán bao gồm các dịch vụ có thể sử dụng lại được với các giao diện rõ ràng, có thể được xuất bản và tương thích với các chuẩn.

2-Các kiến trúc hướng thông điệp: ở đây, các ứng dụng gửi thông điệp qua ESB tới các ứng dụng nhận thông điệp.

3-Các kiến trúc hướng sự kiện: ở đây, các ứng dụng tạo mới và sử dụng các thông điệp độc lập lẫn nhau.

10.3. Phát triển phần mềm bằng kỹ nghệ hướng dịch vụ

10.3.1. Một số mô hình triển khai SOA

Có ba mô hình triển khai chính của SOA là: Service registy (đăng ký dịch vụ), Service broker (môi giới dịch vụ) và Service bus (tuyến dịch vụ).

1-Service registry (mô hình truyền thống để định vị và liên kết các dịch vụ trong một hệ thống SOA)

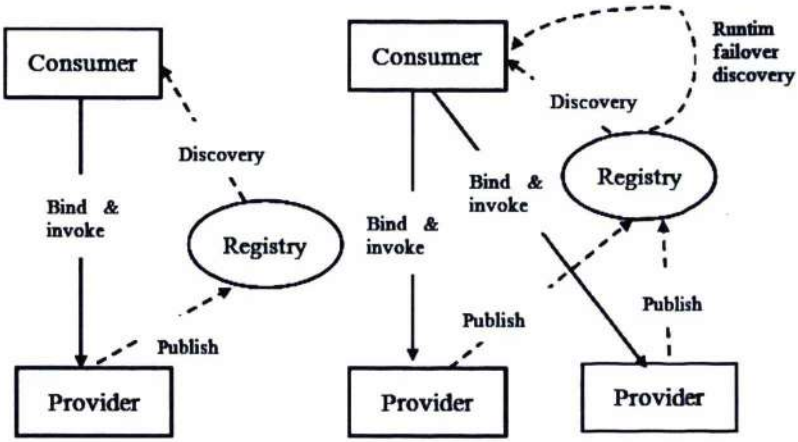
Mô hình đăng ký dịch vụ [22] về cơ bản chỉ cần các chuẩn Web services thông thường là SOAP; UDDI và WSDL (Web Services Description Language). Vấn đề lớn nhất của mô hình này là các liên kết dịch vụ là kết nối tĩnh và phải định nghĩa trong thiết kế, điều này làm cho mô hình trở nên cứng nhắc.

UDDI hỗ trợ nhiều cấu hình khác nhau cho cùng một dịch vụ cung cấp bởi nhiều nhà cung cấp dịch vụ khác nhau. Điều này cho phép chia tải và tăng tính tin cậy bởi vì thư mục dịch vụ (Service directory) có thể tìm kiếm một dịch vụ nào đó trên tất cả các nhà cung cấp dịch vụ hiện có. Đặc điểm chính của SOA là tách rời phần giao tiếp/gọi dịch vụ với phần thực hiện dịch vụ. Điều này có vẻ liên quan đến một công nghệ được đề cập nhiều gần đây là WebService (dịch vụ web). Dịch vụ web cho phép truy cập thông qua định nghĩa giao thức-và-giao tiếp. SOA và dịch vụ web thoạt trông có vẻ giống nhau nhưng chúng không phải là một [9].

Sự khác biệt giữa dịch vụ SOA và dịch vụ Web cũng được coi như sự khác nhau giữa hai yếu tố: khái niệm và thực hiện. Xét trên yếu tố khái niệm, mô hình SOA không nhằm định nghĩa một cách chính xác phương thức dịch vụ được đặc tả để có thể tương tác với nhau. Mô hình SOA sẽ chỉ định nghĩa làm thế nào để các dịch vụ có thể "hiểu" được nhau và từ đó có thể tương tác được với nhau. Dịch vụ Web xây dựng một tập hợp hướng dẫn về thực hiện thông điệp giữa các dịch vụ (thông thường có thể thấy ở thông điệp SOAP khi được chuyển nhận trên HTTP).

Dịch vụ Web cũng là một phương án để thực hiện mô hình SOA, tuy nhiên nó không phải là phương án duy nhất. Bất kỳ phương án thực hiện nào nếu có khả năng xây dựng giao diện dịch vụ trên nền WSDL và giao tiếp với thông điệp XML đều có thể thực hiện mô hình SOA.

Tóm lại, dịch vụ Web là một phương án cụ thể và thực tế phục vụ việc phát triển và triển khai mô hình SOA, là đặc tả công nghệ còn SOA là triết lý thiết kế phần mềm. *Dịch vụ web đưa ra giải pháp kỹ thuật để thực hiện SOA, nhưng SOA cũng có thể thực hiện với các giải pháp kỹ thuật khác không phải dịch vụ web (và không phải tất cả dịch vụ web đều có kiến trúc SOA). Tuy vậy, SOA và dịch vụ web có mối quan hệ tương hỗ: sự phổ biến của dịch vụ web giúp thúc đẩy sự phát triển của SOA, và kiến trúc tốt của SOA sẽ giúp dịch vụ web thành công.*



Hình 10.10. Mô hình Service Registry [22]

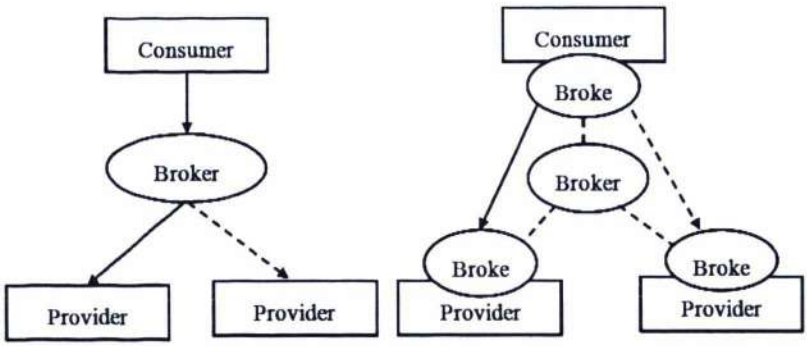
Hình 10.10. Mô hình Service Registry [22]

2-Service broker

Một bộ trung gian làm việc giữa dịch vụ cung cấp và dịch vụ tiêu thụ. Trong mô hình cơ bản, tất cả những thông điệp đều được trung chuyển qua Service broker. Dịch vụ này có thể làm nhiều chức năng như định tuyến dựa trên dữ liệu thông điệp, xử lý lỗi, chuyển đổi thông điệp, chia tải và lọc thông tin. Nó cũng có thể cung cấp dịch vụ bảo mật, chuyển đổi giao thức, lưu vết và các dịch vụ hữu ích khác.

Tuy nhiên, Service broker là nơi có thể xảy ra hiện tượng nghẽn cổ chai và là điểm dễ bị hỏng hóc. Mô hình broker phân tán là một bước cải tiến mới, ở

đó mỗi nền tảng dịch vụ có một broker cục bộ cho phép giao tiếp với một Service broker trung tâm và giao tiếp trực tiếp với các Service broker cùng cấp ở các nền tảng dịch vụ khác.

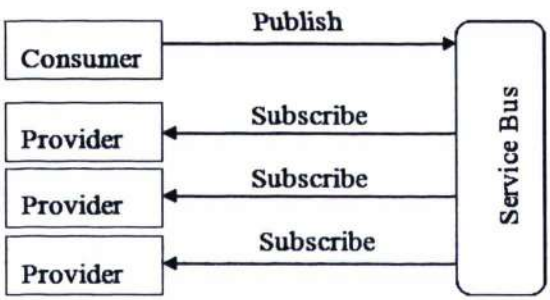


Hình 10.11. Mô hình Service Broker [22]

Hình 10.11. Mô hình Service Broker [22]

3-Service Bus

Service Bus là mô hình ra đời sau nhất trong 3 mô hình nhưng nó đã được sử dụng trong các sản phẩm thương mại quy mô lớn (như IBM, BEA). Service Bus cũng là mô hình có tính kết nối lỏng nhất trong các mô hình, trong đó các dịch vụ không kết nối trực tiếp với nhau. Đôi khi các service bus kết nối với nhau thành một mạng các Service Bus.

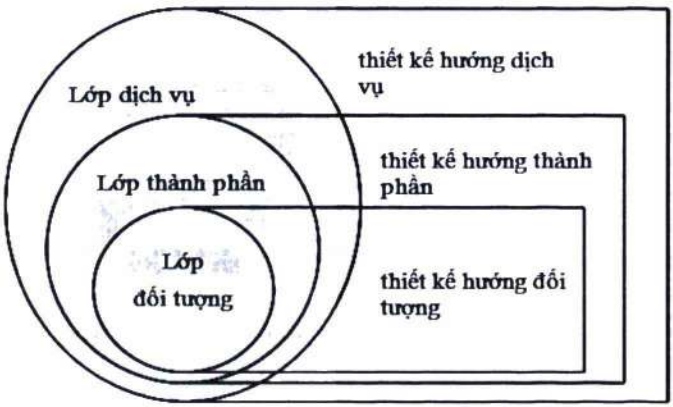


Hình 10.12. Mô hình Service Bus [22]

10.3.2. Các nguyên tắc phát triển ứng dụng theo hướng dịch vụ

Trong sự tiến hoá của các kỹ thuật xây dựng phần mềm, kỹ thuật lập trình hướng đối tượng thích hợp để cài đặt các thành phần. Trong khi các thành phần lại là cách thích hợp nhất để cài đặt các dịch vụ, mặc dù cần hiểu rằng một ứng dụng dựa thành phần tốt không cần thiết là một ứng dụng dịch vụ tốt. Khi vai trò của dịch vụ trong kiến trúc ứng dụng được hiểu rõ, chúng ta có thể tích hợp các thành phần mới và các thành phần hiện có.

Hình 10.13 minh hoạ cách cài đặt ở mức độ đóng gói cao hơn khi tiến gần tới NSD, nó cũng cho thấy dịch vụ là cách thích hợp để thể hiện khung nhìn ngoài của hệ thống với sự tái sử dụng bên trong có sử dụng thiết kế thành phần truyền thống.



Hình 10.13. Các tầng cài đặt trong thiết kế [28]

Việc phát triển các hệ thống dựa trên kỹ nghệ hướng dịch vụ phải tuân theo bốn nguyên tắc sau:

Nguyên tắc 1: Giao diện của dịch vụ phải rõ ràng. Các dịch vụ tương tác qua việc truyền đi các thông điệp tường minh. Chúng ta không cần biết về không gian nằm sau giao diện của dịch vụ. Vượt qua các giao diện của dịch vụ có thể tốn nhiều công sức và chi phí. Các giao diện rõ ràng cho phép cài đặt các tương tác độc lập- nghĩa là không cần biết về nền tảng hay ngôn ngữ lập trình được lựa chọn để cài đặt.

Nguyên tắc 2: Dịch vụ là tự trị. Các dịch vụ hoạt động như là các thực thể độc lập. Không có quyền làm chủ trong một môi trường hướng dịch vụ. Các dịch vụ được triển khai, thay đổi, quản lý một cách độc lập.

Nguyên tắc 3: Các dịch vụ chia sẻ giao diện và giao ước không chia sẻ cài đặt. Các dịch vụ tương tác với nhau chỉ dựa vào giao diện và giao ước sử dụng dịch vụ. Giao ước của dịch vụ mô tả cấu trúc của thông điệp và các ràng buộc giữa các thông điệp, điều này cho phép chúng ta bảo toàn được tính toàn vẹn của dịch vụ. Các giao ước và giao diện phải được duy trì tính ổn định với thời gian. Vì vậy, việc xây dựng chúng một cách mềm dẻo là rất quan trọng.

Nguyên tắc 4: Tính tương thích của dịch vụ phải dựa trên chính sách. Cả người cung cấp và người dùng dịch vụ sẽ phải có các chính sách để tương tác qua các giao diện của dịch vụ. Một minh họa đơn giản về chính sách phía người cung cấp là một dịch vụ có thể đòi hỏi người gọi phải có một tài khoản hợp lệ với người cung cấp dịch vụ. Về phía người dùng dịch vụ, một tổ chức có thể đòi hỏi các lời kêu gọi qua Internet phải được mã hoá.

Có rất nhiều quyết định thiết kế và kiến trúc là nền tảng cho SOA để đạt được các mục tiêu và lợi ích mà kiến trúc này đã nêu ra. Các quyết định kiến trúc thường liên quan tới việc chọn lựa các công nghệ và sản phẩm cần thiết kế để đáp ứng chất lượng cho các thuộc tính dịch vụ được yêu cầu bởi một quy trình nghiệp vụ. Các quyết định thiết kế tập trung vào các lựa chọn dịch vụ sử dụng kỹ thuật được mô tả sau trong phân tích và thiết kế hướng đối tượng. Các quyết định thiết kế và kiến trúc được tiến hành thích hợp với mỗi tầng của SOA (các thành phần của kiến túc hướng dịch vụ).

Trong quá trình thiết kế, có một số quyết định chính:

1- Xác định dịch vụ: các quyết định thiết kế quyết định thành công một kiến trúc hướng dịch vụ.

2- Thiết kế cài đặt thành phần chứa: các quyết định thiết kế và kiến trúc rất quan trọng để một dịch vụ cung cấp các chức năng cho việc mở rộng và bảo trì.

3- Mức độ đóng gói: các lựa chọn thiết kế về dịch vụ phải phù hợp với mức độ tái sử dụng và mềm dẻo được yêu cầu trong ngữ cảnh cụ thể. Các dịch vụ có mức độ đóng gói cao hơn có thể trợ giúp việc đóng gói các thay đổi trong

các dịch vụ kỹ thuật có mức độ đóng gói thấp hơn (các dịch vụ này thường có xu hướng thay đổi thường xuyên hơn so với các nghiệp vụ ở ở mức cao hơn). Các nhân tố của tính mềm dẻo sẽ là các tiêu chuẩn chính cho việc đóng gói hơn là việc chỉ đơn thuần đóng gói chức năng.

4- Tính gắn kết không chặt chẽ và cấu hình lại động là một quyết định thiết kế yêu cầu việc tách giao diện khỏi cài đặt giao thức thông qua các chuẩn mở. Kết nối giữa các thành phần cung cấp và sử dụng dịch vụ cần phải bền vững và được cấu trúc rõ ràng, có khả năng cấu hình lại linh hoạt. Có khả năng cấu hình lại có nghĩa là các thành phần sử dụng dịch vụ và các thành phần cung cấp dịch vụ hiện có thể được lắp ghép lại với các chức năng không bị thay đổi để đạt được các giải pháp nghiệp vụ trong các môi trường kỹ thuật khác nhau và với các ràng buộc thao tác khác nhau của các đối tác kinh doanh mới.

10.3.3. Vòng đời phát triển của dịch vụ

Mối quan tâm chính trong việc xây dựng các dịch vụ theo kiến trúc hướng dịch vụ là các dịch vụ nghiệp vụ cần phải được chuẩn hoá và có mức trừu tượng cao trong xây dựng và triển khai. Cần tập trung chú ý vào quy trình xác định dịch vụ mới có thể bảo đảm việc cài đặt SOA trở lên hiệu quả như mong đợi.

Vòng đời phát triển và định nghĩa dịch vụ cho phép phát triển và triển khai hệ thống SOA hiệu quả.

Hình 10.14.Vòng đời phát triển dịch vụ [28]

Hình 10.14 mô tả vòng đời phát triển dịch vụ, mỗi trạng thái được định nghĩa như sau:

1-Khởi tạo: một dịch vụ tiềm năng được xác định từ việc phân tích các yêu cầu của doanh nghiệp từ trên xuống (top-down) và việc mô hình hoá quy trình nghiệp vụ.

2-Thu nhận: đội ngũ kiến trúc dịch vụ thông báo là đã nhận được yêu cầu dịch vụ và bắt đầu đánh giá nó.

3-Chứng nhận: đội ngũ kiến trúc dịch vụ đã đánh giá dịch vụ và thống nhất giao diện chức năng của nó.

4-Phân loại: dịch vụ được xác định và được chuyển giao cho đội phát triển.

5-Cài đặt: đội ngũ phát triển dịch vụ cài đặt dịch vụ theo các quy tắc và hướng dẫn phát triển được định nghĩa bởi framework.

6-Kiểm thử: đội ngũ kiểm thử dịch vụ kiểm chứng tính năng cũng như các đặc tính chất lượng của dịch vụ.

7-Xuất bản: dịch vụ được xuất bản để có thể được sử dụng bởi các dịch vụ khác hoặc các ứng dụng định hướng quy trình.

10.3.4. Các pha cơ bản xây dựng hệ thống SOA

Hiện nay chưa có một quy trình cụ thể để phát triển các ứng dụng theo kiến trúc hướng dịch vụ, tuy nhiên, dựa trên thực tế, một số bước sau đã được đưa ra nhằm tham khảo quyết định phát triển hệ thống.

Xây dựng hệ thống SOA có thể trải qua 6 bước (6 pha) sau đây [22]:

- *Bước 1*: Phân tích hướng dịch vụ (Service oriented analysis). Đây là giai đoạn đầu để quyết định phạm vi của hệ thống SOA. Hệ thống gồm những dịch vụ nào? Tầng dịch vụ được lược đồ hoá ra (mapped out) và chia dịch vụ ra thành các mô hình, bao gồm hệ thống SOA sơ bộ.

- *Bước 2:* Thiết kế hướng dịch vụ (Service oriented design). Đây là giai đoạn có sự kết hợp chặt chẽ về sự thỏa hiệp của doanh nghiệp và nguyên lý hướng dịch vụ thành quy trình thiết kế dịch vụ. Trong pha này, làm cho người thiết kế dịch vụ phải đương đầu với giải quyết vấn đề then chốt đó là thiết lập

nên những ranh giới thông qua các dịch vụ. Các tầng dịch vụ được thiết kế trong giai đoạn này có thể bao gồm tầng *phổ khúc (orchestrantion)*, các kết quả của nó là trong sự xác định quy trình nghiệp vụ hình thức.

- *Bước 3:* Phát triển dịch vụ (Service development). Trong bước này là giai đoạn xây dựng thực tế. Ở đây vấn đề về nền tảng phát triển đi vào hoạt động, không quan tâm tới nó là loại dịch vụ nào. Một cách cụ thể, là sự lựa chọn ngôn ngữ lập trình, môi trường phát triển sẽ quyết định những mẫu dịch vụ và quy trình nghiệp vụ tầng phổ khúc nào phù hợp với thiết kế.

- *Bước 4:* Kiểm thử dịch vụ (Service testing). Để đưa ra những tiềm năng cho việc dùng lại và bao gồm cả những trạng thái không biết trước được, các dịch vụ được yêu cầu trải qua được sự nghiêm ngặt của việc kiểm thử trước khi được triển khai thành các sản phẩm.

- *Bước 5:* Triển khai dịch vụ (Service deployment). Giai đoạn thực thi này đưa đến việc cài đặt và cấu hình cho các thành phần phân tán, các giao diện dịch vụ và nhiều sản phẩm trung gian (middlewareproducts) kết hợp với nhau thành những máy chủ (Server).

- *Bước 6:* Quản trị dịch vụ (Service administration). Sau khi các dịch vụ được triển khai, vấn đề quản lý các ứng dụng trở thành hàng đầu, mối quan tâm cho hệ thống phân tán, các ứng dụng dựa trên các thành phần (Component based applications) và việc xem xét chúng như các dịch vụ trong một tổng thể.

Các bước trên có thể chia nhỏ hơn nữa tùy thuộc vào độ phức tạp của bài toán. Đó chưa phải là quy trình bắt buộc xây dựng một dự án SOA thành công. Trong một số trường hợp, chúng ta cần thêm vào hoặc loại bỏ đi một số bước để phù hợp với yêu cầu cụ thể.

Kiến trúc hướng dịch vụ được đưa ra nhằm loại bỏ sự trùng lặp và dư thừa qua việc tái sử dụng và tích hợp. Cách đơn giản nhất là để bắt đầu với SOA là thử với một dịch vụ mà chúng ta biết rằng có nhiều cài đặt trong nhiều ứng dụng khác nhau, sau đó bắt đầu xây dựng kế hoạch và chiến lược để loại bỏ các dịch vụ dư thừa. Quy trình cài đặt này sẽ giúp chúng ta đáp ứng được các yêu cầu cơ bản của tổ chức ẩn sau bước chuyển đổi thành công sang SOA.

Khi chúng ta đã thực hiện được quy trình này, chúng ta sẽ có các công cụ và hiểu biết để mở rộng quy mô áp dụng SOA trong tổ chức của mình.

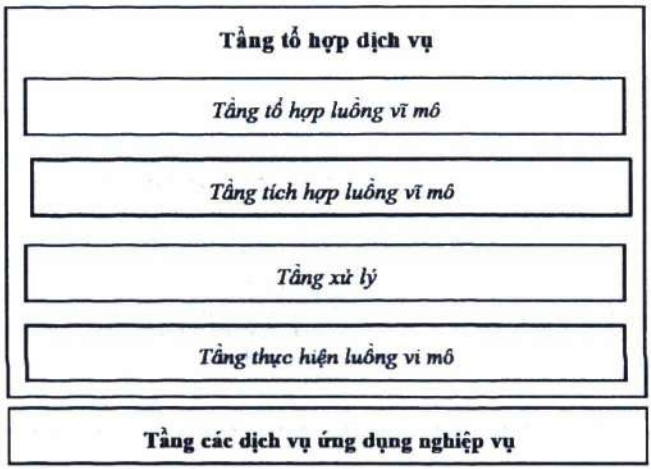
10.3.5. Thiết kế và thực hiện luồng công việc

Một trong vấn đề chính của công nghệ SOA là tạo ra các mẫu (pattern) và sử dụng trong các ứng dụng. Trong ngôn ngữ của mẫu, các mẫu vĩ mô - Macroflow thiết lập các khuôn cảnh (kịch bản) và xếp đặt các khái niệm cơ sở cho kiến trúc tổng thể. Mẫu của kiến trúc tích hợp dựa trên quá trình mô tả cách để thiết kế một kiến trúc dựa trên lớp con cho tầng tổng hợp dịch vụ, chính là mẫu khái niệm Macromicroflow [14].

Mô hình kiến trúc của SOA được tổ chức thành hai tầng chính như sau:

1-Tầng tổ hợp dịch vụ;

2-Tầng các dịch vụ ứng dụng nghiệp vụ.



Hình 10.15. Cấu trúc phân cấp của SOA [9]

Tầng tổ hợp dịch vụ (Service Composition Layer) được phân chia thành 4 tầng con: tầng tổ hợp luồng vĩ mô (Macroflow Composition), lớp tích hợp luồng vĩ mô (Macroflow Integration), tầng xử lý (Dispatching) và tầng thực hiện luồng vi mô (Microflow Execution).

Trên tầng tổ hợp có 2 luồng:

1-Luồng xuyên suốt (Long-running Flow), một luồng mô tả định hướng nghiệp vụ của doanh nghiệp, được gọi là luồng vĩ mô (*macroflow);*

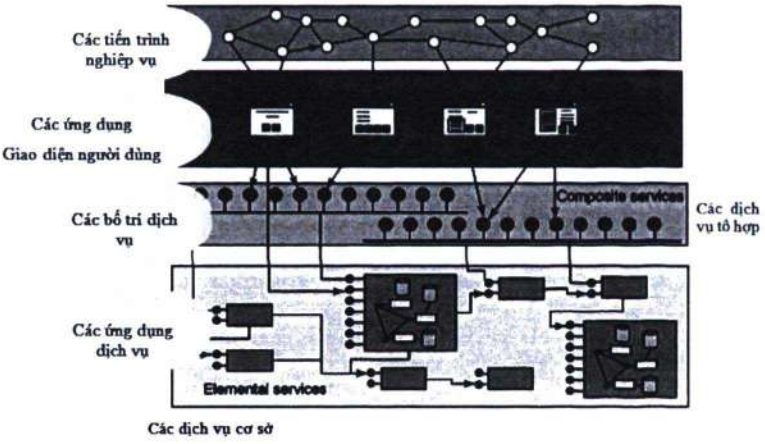
2-Luồng giao dịch ngắn (Short-transactional Flow), mô tả các quy trình nghiệp vụ được tin học hóa, được gọi là luồng vi mô (*microflow).*

Cả hai macroflows và microflows đều có thể được làm mịn dần. Sự khác nhau của các loại làm mịn này ở chỗ sử dụng quá trình làm mịn các luồng qui trình nguyên thủy. Qui trình nguyên thủy là mẫu nguyên thủy tổng quát có thể sử dụng để mô hình hoá tất cả các loại tiến trình công việc.

Các mẫu nguyên thủy cho phép chúng ta phát triển mô hình nghiệp vụ dựa trên khái niệm kiến trúc tiến trình hướng thành phần. Các mẫu nguyên thủy UML 2.0 có thể sử dụng để mô hình hoá macroflows và microflows.

Mỗi thành phần tiến trình được làm mịn dần bằng cách sử dụng mô hình macroflows cho đến khi biểu diễn được cho những hoạt động cơ bản nhất và cần thiết trong mô hình.

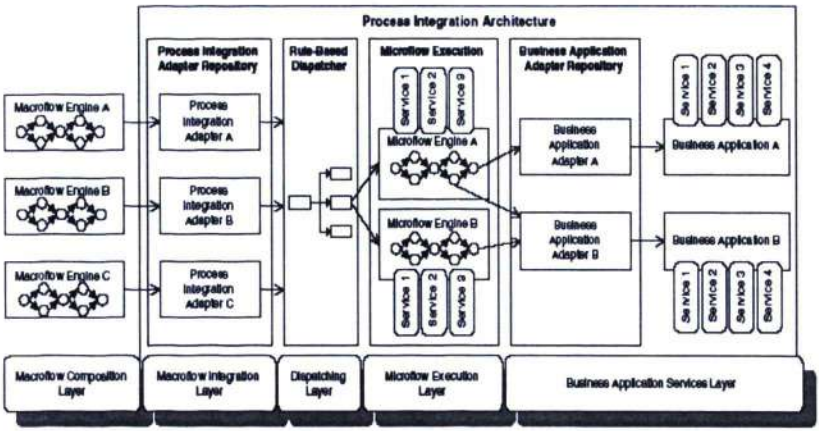
Với kiến trúc này, các phần tử của SOA có thể biểu diễn tổng thể như sau:



Hình 10.16. Các phần tử của SOA [9]

Hình 10.17 thể hiện một cấu hình của kiến trúc tích hợp dựa vào quá trình, trong đó nhiều macroflow engines thực hiện các macroflows. Process-integration adapters được sử dụng để tích hợp các luồng công việc vĩ mô theo

các phương diện kỹ. Tầng xử lý cho phép mở rộng số lượng các máy xử lý vi mô (microflow engines).



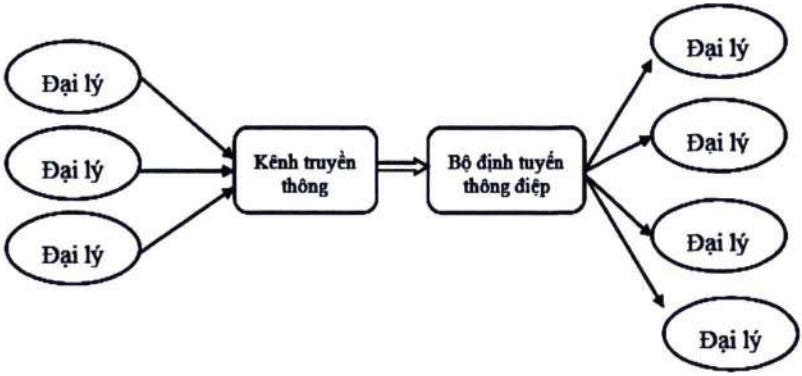
Hình 10.17. Minh họa về một cấu hình kiến trúc dựa vào quá trình [14]

Oracle đã thông báo về việc phổ biến rộng rãi bộ sản phẩm *Oracle® SOA Suite* - bộ sản phẩm lớp giữa dựa trên các chuẩn, sử dụng nhằm mục đích giúp dễ dàng xây dựng, ứng dụng và quản lý kiến trúc hướng dịch vụ [22].

Những tiền đề cơ bản của một hệ thống kiến trúc hướng dịch vụ là phân tách các ứng dụng ra khỏi các ứng dụng khác và từ đó, các ứng dụng được tự trị trong điều kiện riêng [14]. Vì vậy, khi mỗi ứng dụng khác nhau trong một tổ chức có nhu cầu trao đổi thông tin với nhau (tích hợp ứng dụng) thì họ phải giao tiếp theo một quy trình tiêu chuẩn. Việc trao đổi thông tin này được thực hiện thông qua các thông điệp (Message) được trao đổi theo qui định của doanh nghiệp. Các thông điệp này thường sử dụng hiện nay là thông điệp XML. Đối với trường hợp của một kiến trúc hướng dịch vụ rất đơn giản, chúng ta xem xét các tin nhắn từ XML của nó nhiều hơn hoặc ít hơn các mẫu sự kiện trong giao tiếp hàng ngày.

Các thông điệp này phải được gửi cho các đơn vị và đây là nơi thực hiện các nhiệm vụ phát sinh. Các nhiệm vụ được gửi dưới dạng tin nhắn XML đơn giản và các ứng dụng sẽ nhận được để xử lý.

Hình 10.18 mô tả sự trao đổi thông điệp giữa các đại lý được giao thực hiện các dịch vụ tương ứng qua các qua kênh truyền thông. Các thông điệp được chuyển đến các điểm đến thích hợp thông qua bộ định tuyến thông điệp (Message Router). Đại lý có sẵn cho mỗi ứng dụng, duy nhất chỉ có nhiệm vụ thực hiện các đại lý sẽ được cung cấp đầu vào cho NSD, cho các kênh truyền tin và cũng có thể tiếp nhận được tin nhắn đến của các ứng dụng.



Hình 10.18. Một khung kiến trúc SOA

Các đại lý là đại diện của các ứng dụng, có thể cung cấp các thông điệp được đưa vào hàng đợi chuẩn. Tất cả các đại lý bên trái thực hiện các dịch vụ của chương trình ứng dụng và gửi kết quả (thông điệp XML) vào hàng đợi.

Các kênh truyền thông sẽ chịu trách nhiệm xử lý các hàng đợi (queues). Ngoài ra, các kênh truyền còn chịu trách nhiệm về độ tin cậy của các thông điệp và bảo đảm thực hiện theo mức độ ưu tiên của các dịch vụ. Chúng bảo đảm rằng những tin nhắn đã nhận được là bản gốc. Thông tin trao đổi với nhau trong mô hình kiến trúc dịch vụ hướng tới đơn giản bằng cách sử dụng XML. Tin nhắn nhanh chóng được chuyển giao cho bộ định tuyến (router).

Các thông điệp trên các kênh có thể được chuyển đổi sang định dạng khác nhau. Sự chuyển đổi này phải tuân theo một sơ đồ cụ thể dựa trên các quy tắc kinh doanh. Hơn nữa, điều đó thường dễ dàng cho các đại lý để gửi tài liệu XML thô và sau đó được chuyển đổi thông điệp thích hợp của SOAP.

Chẳng hạn, một đại lý có thể là một trang dịch vụ web phục vụ gửi tin nhắn SOAP, có thể đưa tin nhắn vào hàng đợi và các kênh thông báo không còn trách nhiệm chỉ đọc tin nhắn và kiểm tra hiệu lực của nó theo sơ đồ kinh doanh cơ sở quy định.

Nghiên cứu khung công việc (Framework), chúng ta có thể áp dụng một kịch bản đơn giản, khi đã nhận được thông báo được gửi từ ứng dụng X tới cho ứng dụng Y [14]:

- Ứng dụng X đã có các đại lý cung cấp thông tin, là một dịch vụ web. Một thông báo được ứng dụng X gửi cho các dịch vụ web. Các dịch vụ web chuyển những tin nhắn này vào hàng đợi;

- Kênh truyền thông (Channel) đọc các tài liệu thô XML từ hàng đợi. Nó kiểm tra nếu phù hợp với mẫu XML thì chuyển thông tin này tới *bộ định tuyến (router)*. Kênh truyền thông có thể được triển khai thực hiện như dịch vụ Window và theo dõi các dịch vụ của hàng đợi;

- Router có thể dễ dàng tổ chức để có thể được sử dụng bởi các kênh. Router khi nhận được tin nhắn sẽ tìm trong bộ nhớ cache các biểu thức chính quy Xpath được ánh xạ tới một số cuối cụ thể (endpoints). Nếu được thông báo được đánh giá phù hợp thì sẽ gửi đến các đại lý của ứng dụng Y thông qua các yêu cầu (invocation) động của các hiệp hội hoặc thậm chí chung thông qua web proxy;

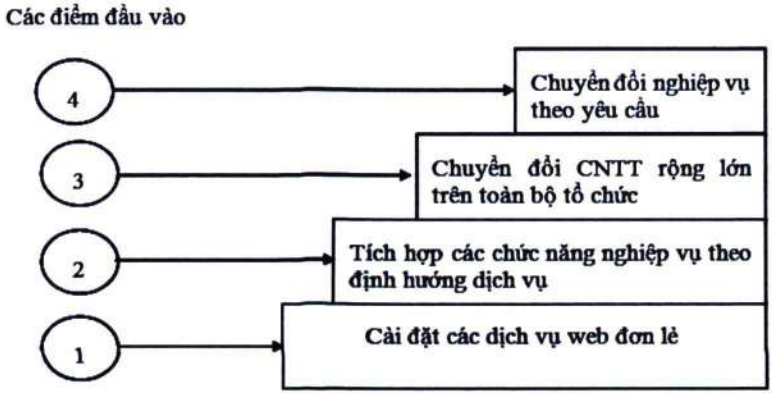
- Ứng dụng Y có thông tin XML, nó có thể phân tích cú pháp và thực hiện các công việc theo yêu cầu. Cache có thể được cài đặt như là một bảng trong CSDL hoặc một file tin đơn giản;

Xét kịch bản có một số lựa chọn khác nhau trong khung công việc nêu trên. Đó có thể là một tin nhắn SOAP hoàn toàn thông qua các khuôn khổ công việc. Có thể sử dụng các đại lý mà không cần phải sử dụng dịch vụ web và có thể áp dụng cho bất kỳ loại ứng dụng nào. Những kỹ thuật này có thể được áp dụng theo công nghệ WSE 2,0 mới nhất. Vì vậy SOAP có thể được gửi tin nhắn không chỉ qua HTTP, nhưng cũng có thể trên TCP.

10.3.6. Các mức độ thực hiện kiến trúc hướng dịch vụ

Việc sử dụng kiến trúc hướng dịch vụ không bị giới hạn cho các tổ chức lớn. Trong thực tế, kiến trúc này tạo ra cơ hội cho các tổ chức vừa và nhỏ. Giả sử, các doanh nghiệp nhỏ có thể sử dụng dịch vụ là việc nhận các hoá đơn: chúng ta có thể dùng dịch vụ mạng để tạo ra một dịch vụ nhận hoá đơn. Các hoá đơn này sẽ được lưu trữ bởi dịch vụ cho đến khi hệ thống kế toán trên máy tính cá nhân của người dùng kết nối đến nó để nhận hoá đơn về bằng cách sử dụng dịch vụ mạng. Các hoá đơn sau đó có thể cập nhận một cách tự động trên máy tính cá nhân,... Bằng cách này, bất kỳ tổ chức nào, không phụ thuộc vào quy mô lớn hay nhỏ, đều có thể nhận được lợi ích từ việc sử dụng kiến trúc hướng dịch vụ.

Một tổ chức có thể dùng nhiều cách khác nhau để tiếp nhận được kiến trúc hướng dịch vụ, tuỳ thuộc vào mục tiêu và các ràng buộc CNTT.



Hình 10.19. Các mức hực hiện kiến trúc hướng dịch vụ [28]

Ghi chú:

- Đầu vào 1: Tạo các dịch vụ từ các nhiệm vụ trong các ứng dụng mới và cũ;

- Đầu vào 2: Tích hợp các dịch vụ từ nhiều ứng dụng bên trong và bên ngoài tổ chức để thực hiện một nục đích nghiệp vụ;

- Đầu vào 3: Một cài đặt được kiến trúc cho phép tích hợp nhiều chức năng nghiệp vụ trong toàn bộ một tổ chức;

- Đầu vào 4: Chuyển đổi rộng khắp các mô hình nghiệp vụ hiện có hoặc triển khai các mô hình nghiệp vụ mới

Các mức thực hiện kiến trúc hướng dịch vụ bao gồm:

Mức 1: Cài đặt các dịch vụ riêng lẻ

Mức này tạo ra các dịch vụ từ các nhiệm vụ có trong các ứng dụng mới hoặc các ứng dụng hiện có. Kiến trúc hướng dịch vụ đem lại khả năng thiết kế các ứng dụng và các hệ thống cung cấp dịch vụ cho các ứng dụng khác thông qua các giao diện được xuất bản. Cách tạo ra các ứng dụng này có thể đưa đến một mô hình lập trình mạnh hơn, linh hoạt hơn, giảm chi phí cả trong phát triển và bảo trì.

Mức 2: Tích hợp hướng dịch vụ các chức năng nghiệp vụ

Mức tiếp theo là tích hợp hướng dịch vụ. Bước này bao gồm việc tích hợp các dịch vụ từ nhiều ứng dụng khác nhau cả bên trong lẫn bên ngoài doanh nghiệp để phục vụ cho mục đích nghiệp vụ. Mức độ này cũng phải hỗ trợ nhiều kiểu tích hợp, bao gồm:

- *Tích hợp ứng dụng:* gồm cả sự phát triển các giao diện mới để thể hiện các ứng dụng hiện có;

- *Tích hợp thông tin:* gồm cả dữ liệu doanh nghiệp và dữ liệu phòng ban;

- *Tích hợp giao diện:* gồm tích hợp qua sự tích hợp, sắp xếp các ứng dụng và dịch vụ với nhiều giao diện;

- *Tích hợp hệ thống:* gồm sự kết nối các hệ thống không đồng nhất, các hệ thống hiện có và tích hợp ứng dụng.

Mức 3: Chuyển đổi doanh nghiệp CNTT có quy mô lớn.

Mức cài đặt SOA rộng hơn là một cài đặt được kiến trúc hoá cho phép tích hợp nhiều chức năng nghiệp vụ trong toàn doanh nghiệp.

Mức 4: Chuyển đổi nghiệp vụ theo yêu cầu.

Mức 4 là mức cuối cùng trong phát triển theo kiến trúc hướng dịch vụ. Mức này là định hướng chiến lược, hướng tới việc chuyển đổi mở rộng của các mô hình nghiệp vụ hiện có hoặc việc triển khai của các mô hình nghiệp vụ mới.

TÓM TẮT CHƯƠNG

SOA là "*Khái niệm về hệ thống trong đó mỗi ứng dụng được xem như một nguồn cung cấp dịch vụ*".

Điểm khác biệt chính của SOA so với các mô hình có kiến trúc tương tự là cung cấp khả năng giao tiếp giữa các thành phần trong hệ thống sử dụng thông điệp dựa trên các chuẩn giao tiếp đã được chuẩn hóa. Chính nhờ đặc điểm này, hệ thống SOA trở nên độc lập với nền tảng. Các dịch vụ hoạt động trên nền các nền tảng khác nhau vẫn có thể giao tiếp với nhau nhờ vào các giao diện giao tiếp đã được chuẩn hóa để cộng tác xử lý một tác vụ nào đó.

Các thành phần dùng để cung cấp các chức năng của dịch vụ. Các thành phần sử dụng dịch vụ, thành phần cung cấp dịch vụ và thành phần môi giới dịch vụ tại nơi đăng ký dịch vụ đều được định nghĩa và mô tả rõ các dịch vụ.

Dịch vụ là khái niệm then chốt trong SOA. Có thể hiểu dịch vụ như là một chức năng (mô-đun) thực hiện qui trình nghiệp vụ nào đó. Nó được xác định rõ ràng, tự chứa, và không phụ thuộc vào ngữ cảnh hoặc tình trạng của các dịch vụ khác.

Một cách cơ bản, SOA là tập hợp các dịch vụ kết nối "mềm dẻo" với nhau (nghĩa là một ứng dụng có thể "nói chuyện" với một ứng dụng khác mà không cần biết các chi tiết kỹ thuật bên trong), có giao tiếp (dùng để gọi hàm dịch vụ) được định nghĩa rõ ràng và độc lập với nền tảng hệ thống, và có thể tái sử dụng. SOA là cấp độ cao hơn của phát triển ứng dụng, chú trọng đến qui trình nghiệp vụ và dùng giao tiếp chuẩn để giúp che đi sự phức tạp kỹ thuật bên dưới.

Các dịch vụ trong SOA có đặc điểm rất quan trọng như: Các dịch vụ là có thể tìm kiếm được; Các dịch vụ có tính liên thông; Các dịch vụ không được gắn kết chặt chẽ với nhau; Các dịch vụ là phức hợp, bao gồm nhiều thành phần, được đóng gói ở mức cao; Các dịch vụ trong suốt về vị trí; Các dịch vụ có khả năng tự hàn gắn.

Thiết kế SOA tách riêng phần thực hiện dịch vụ (phần mềm) với giao tiếp gọi dịch vụ. Điều này tạo nên một giao tiếp nhất quán cho ứng dụng khách (Client) sử dụng dịch vụ bất chấp công nghệ thực hiện dịch vụ.

Vì ưu điểm quan trọng nhất của SOA là khả năng kết nối "mềm dẻo" (nhờ sự chuẩn hóa giao tiếp) *và tái sử dụng* nên các dịch vụ có thể được sử dụng với trình Client chạy trên nền tảng bất kỳ và được viết với ngôn ngữ bất kỳ. Việc thiết kế dịch vụ phải tuân theo những nguyên tắc nhất định.

SOA được xây dựng dựa trên các chuẩn rất phổ biến là SOAP và XML. Hai chuẩn này đóng vai trò là thành phần xương sống để truyền nhận các thông điệp giữa các đối tượng truyền thông.

Có ba mô hình triển khai chính của SOA là: Service registy (đăng ký dịch vụ), Service broker (môi giới dịch vụ) và service bus (tuyến dịch vụ).

Việc phát triển các hệ thống dựa trên kỹ nghệ hướng dịch vụ phải tuân theo một số nguyên tắc nhất định.

Mối quan tâm chính trong việc xây dựng các dịch vụ theo kiến trúc hướng dịch vụ là các dịch vụ nghiệp vụ cần phải được chuẩn hoá và có mức trừu tượng cao trong xây dựng và triển khai. Cần tập trung chú ý vào quy trình xác định dịch vụ mới có thể bảo đảm việc cài đặt SOA trở lên hiệu quả như mong đợi.

Vòng đời phát triển và định nghĩa dịch vụ cho phép phát triển và triển khai hệ thống SOA hiệu quả.

Hiện nay chưa có một quy trình cụ thể để phát triển các ứng dụng theo kiến trúc hướng dịch vụ, tuy nhiên, khi triển khai cần đưa ra một số bước quyết định phát triển hệ thống.

CÂU HỎI CHƯƠNG 10

10.1. Điểm khác biệt chính của SOA so với các mô hình có kiến trúc tương tự?

10.2. Các *tầng của SOA* dùng để biểu diễn cái gì?

10.3. Sơ lược về các thành phần của một kiến trúc hướng dịch vụ?

10.4. Hệ thống SOA có các tính chất đặc biệt gì?

10.5. Có thể hiểu dịch vụ như thế nào?

10.6. Các dịch vụ trong SOA có những đặc điểm gì?

10.7. Tư tưởng chính của thiết kế SOA là muốn tách cái gì với cai gì? điều này giúp ích gì cho ứng dụng phía khách muốn sử dụng dịch vụ?

10.8. Nhờ những ưu điểm quan trọng nào của SOA mà các dịch vụ có thể được sử dụng với trình Client chạy trên nền tảng bất kỳ và được viết với ngôn ngữ bất kỳ?

10.9. Tại sao chúng ta phải quan tâm đến vòng đời phát triển hệ thống SOA?

10.10. Khi xây dựng một tiến trình cụ thể để phát triển các ứng dụng theo kiến trúc hướng dịch vụ, thì Anh/Chị sẽ làm gì trong pha đầu tiên?

TÀI LIỆU THAM KHẢO

A-TIẾNG VIỆT

[1]. Nguyễn Văn Ba (2003), *Phân tích và thiết kế hệ thống thông tin -Các phương pháp có cấu trúc*, Nhà xuất bản Đại học quốc gia Hà Nội.

[2]. Đoàn Văn Ban, Nguyễn Thị Tĩnh (2011), *Giáo trình Phân tích thiết kế hệ thống hướng đối tượng bằng UML*, Nhà xuất bản Đại học Sư phạm.

[3]. Collongues A., J.Hugues, B.Laroche, Meris (1994), *Phương pháp thiết kế hệ thống thông tin tin học hóa phục vụ quản lý doanh nghiệp*, Nhà xuất bản Khoa học kỹ thuật, Bản dịch của Trương Văn Tú.

[4].Thạc Bình Cường, Nguyễn Đức Mận (2011), *Kiểm thử và đảm bảo chất lượng phần mềm*, Nhà xuất bản Bách khoa, Hà Nội.

[5]. Đặng Văn Đức (2002), *Phân tích thiết kế hướng đối tượng bằng UML (Thực hành với Rational Rose)*, NXB Khoa học và Kỹ thuật, Hà Nội.

[6]. Nguyễn Xuân Huy (1996), *Giáo trình Công nghệ phần mềm*, Đại học Tổng hợp TP. Hồ Chí Minh.

[7]. Đỗ Xuân Lôi (1999), *Cấu trúc dữ liệu và giải thuật*, tái bản lần thứ 5, Nhà xuất bản Thống kê.

[8]. Microsoft (1999), *Networking Essentials (Mạng căn bản)*, Bản tiếng Việt của Ban Biên dịch VN-GUIDE, Nhà xuất bản Thống kê.

[9]. Lê Văn Phùng (2014), *Kỹ nghệ phần mềm*, Nhà xuất bản Thông tin và Truyền thông, tái bản lần 1.

[10]. Lê Văn Phùng (2014), *Kỹ thuật phân tích và thiết kế hệ thống thông tin hướng cấu trúc*, Nhà xuất bản Thông tin và Truyền thông, tái bản lần 3.

[11]. Lê Văn Phùng (2014), *Các mô hình cơ bản trong phân tích và thiết kế hướng đối tượng*, Nhà xuất bản Thông tin và Truyền thông, tái bản lần 1.

[12]. Lê Văn Phùng, Lê Hương Giang (2015), *Kỹ nghệ phần mềm nâng cao*, Nhà xuất bản Thông tin và Truyền thông, tái bản lần 1.

[13]. Lê Văn Phùng (2016), *Hệ thống thông tin quản lý*, Nhà xuất bản Thông tin và Truyền thông, tái bản lần 1.

[14]. Viện CNTT, Viện Hàn lâm Khoa học và Công nghệ Việt Nam, *Nghiên cứu ứng dụng kiến trúc phần mềm hướng dịch vụ trong quản lý hành chính nhà nước*, Đề tài CSCL'08.01, Doãn Ngọc Liên (chủ nhiệm), Hà Nội, 2008.

[15]. Viện CNTT, Viện Hàn lâm Khoa học và Công nghệ Việt Nam (2009), *Nghiên cứu điện toán đám mây và ứng dụng*, Đề tài cơ sở, Hà Nội.

[16]. Nguyễn Văn Vy, Nguyễn Việt Hà, *Giáo trình kỹ nghệ phần mềm*, Nhà xuất bản Giáo dục Việt Nam, 2009.

[17]. Nguyễn Văn Vy, *Giáo trình phân tích thiết kế các hệ thống thông tin*. NXB Giáo dục Việt Nam, 2010.

B. TIẾNG ANH

[18]. A. Alain Ph.D (1999), Managing Software Product Quality Evaluation with ISO Standards. Software Engineering Management Research Lab. Universite du Quebec à Montreal, Canada in Cooperation with GAIA and INFOJED Vietnam.

[19]. L. Bass, P.Clements, Kazman, Software Architecture in Practice, Addition-Wesley, 1998.

[20]. J. Chikofsky and J.H. Cross, Reverse Engineering and Design Recovery: A Taxonomy, vol 7, pp 13-17, January 1990.

[21]. D. Garlan and M. Shaw, An instruction to Software Architecture, Advances in Software Engineering and Klowlege Engineering, Addion-Wesley, 1993.

[22]. IBM Red Book Team (2004), Pattern: Implementing an SOA using Enterprise Service Bus, IBM.

[23]. Roger S. Pressman (2001), A Practitioner's Approach, McGraw Hill, Fifth Edition.

[24]. Richard Koch (1997), The 80/20 principle: The secret of achieving more with less. Nicholas Brealey Publishing, London.

[25]. I. Sommerville (2001), Software Engineering. 6th Edition, Addison-Wasley.

[26]. F. Sorensen (2004), A Comparison of Distributed Object Technologies.

C. INTERNET

[27].Https://tailieu.vn/doc/bai-giang-he-thong-dieu-khien-phan-tan-phan-1-1841935.html 2016: Hoàng Minh Sơn (2005), Bài giảng Hệ thống điều khiển phân tán, Đại học Bách khoa Hà Nội.

[28].Https://www.slideshare.net/ThuyetNguyen11/kin-trchngdchvwebservice-2016: Kiến trúc hướng dịch vụ, Khoa CNTT, ĐHSP KT Hưng Yên.

[29].Http://doc.edu.vn/tai-lieu/de-tai-kien-truc-huong-dich-vu-va-ung-dung-service-oriented-architecture-23969/ 2016. TAI LIEU - EBOOK: Kiến trúc hướng dịch vụ và ứng dụng (2013).

NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN
Địa chỉ: Phường Tân Thịnh - thành phố Thái Nguyên - tỉnh Thái Nguyên
Điện thoại: 0208.3840.018 - Fax: 0208.3840.017
Website: nxb.tnu.edu.vn * E.mail: nxb.dhtn@gmail.com

LÊ VĂN PHÙNG - NGUYỄN VĂN TẢO

GIÁO TRÌNH CÔNG NGHỆ PHẦN MỀM NÂNG CAO

Chịu trách nhiệm xuất bản:
TS PHẠM QUỐC TUẤN
Phó Giám đốc - Phụ trách

Chịu trách nhiệm nội dung
PGS.TS NGUYỄN ĐỨC HẠNH
Tổng biên tập

Biên tập:	PHẠM VĂN VŨ
Thiết kế bìa:	LÊ THÀNH NGUYÊN
Trình bày:	HOÀNG ĐỨC NGUYÊN
Sửa bản in:	NGUYỄN THỊ THUỲ DƯƠNG

Liên kết xuất bản:
Trường Đại học Công nghệ Thông tin và Truyền Thông - Đại học Thái Nguyên
Địa chỉ: Xã Quyết Thắng - thành phố Thái Nguyên - tỉnh Thái Nguyên

ISBN: 978-604-915-652-6

In 300 cuốn, khổ 17 x 24 cm, tại Xưởng in - Nhà xuất bản Đại học Thái Nguyên (Địa chỉ: phường Tân Thịnh, thành phố Thái Nguyên). Giấy phép xuất bản số: 2934-2018/CXBIPH/04-105/ĐHTN. Quyết định xuất bản số: 24/QĐ-NXBĐHTN. In xong và nộp lưu chiểu quý I năm 2019.